NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENTO DE INFORMÁTICA

FREDERICO ALBERTO FRANCO PINHEIRO

Licenciado em Engenharia Informática

SIMULAÇÃO DE FLUIDOS EM TEMPO REAL COM INTERAÇÃO COM CORPOS RÍGIDOS

REAL-TIME FLUID SIMULATION WITH INTERACTION WITH RIGID-BODIES

MESTRADO EM ENGENHARIA INFORMÁTICA

Universidade NOVA de Lisboa
Maio, 2023

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTAMENTO
DE INFORMÁTICA

SIMULAÇÃO DE FLUIDOS EM TEMPO REAL COM INTERAÇÃO COM CORPOS RÍGIDOS

REAL-TIME FLUID SIMULATION WITH INTERACTION WITH RIGID-BODIES

FREDERICO ALBERTO FRANCO PINHEIRO

Licenciado em Engenharia Informática

Orientador: Rui Nóbrega
Professor Auxiliar, NOVA School of Science and Technology

Júri:

Presidente: João Carlos Gomes Moura Pires
Professor Associado, NOVA School of Science and Technology

Arguente: António Ramires Fernandes
Professor Auxiliar, Universidade do Minho

Orientador: Rui Pedro da Silva Nóbrega
Professor Auxiliar, NOVA School of Science and Technology

MESTRADO EM ENGENHARIA INFORMÁTICA

Universidade NOVA de Lisboa
Maio, 2023

Simulação de fluidos em tempo real com interação com corpos rígidos

Agradecimentos

Ao meu orientador, Rui Nóbrega, por ter acreditado e investido em mim, mostrando o apoio e disponibilidade para me guiar durante todo este processo.

À minha mãe, ao meu pai e irmã, que sempre estiveram ao meu lado para me apoiar e investir em toda a minha carreira académica, e principalmente, por sempre acreditarem em mim.

Aos meus familiares e amigos, que também estiveram sempre presentes para me apoiar e ajudar a superar dificuldades, tornando esta experiência muito mais memorável.

Aos animais de estimação que passaram na minha vida, que sempre estiveram prontos para me ouvir.

Aos criadores dos jogos que me apaixonam, por me mostrarem algo pelo qual lutar e me inspirarem a seguir a minha carreira de sonho.

E por último, a mim, por nunca desistir, e principalmente, por acreditar que os sonhos são possíveis.

"There must be a beginning of any great matter, but the continuing unto the end until it be thoroughly finished yields the true glory." (Francis Drake)

Resumo

Simulação de fluidos é composta por um conjunto de técnicas e ferramentas que visam emular os comportamentos dos fluidos em diferentes cenários. Hoje em dia, existe uma vasta diversidade de aplicações que recorrem a esta classe de simulações, podendo ser usadas para investigações científicas e no entretenimento, como filmes e videojogos. Deste modo, simulações como estas, podem representar vários tipos de fluido, tais como: rios, oceanos, ventos, fluxos de corrente, ou lava. Como tal, para se proceder à criação de fluidos, têm de ser considerados diferentes aspetos do seu comportamento, como a velocidade, a densidade do fluido, a pressão do fluido ou o nível de viscosidade do mesmo.

Nesta dissertação pretende-se investigar técnicas de simulação de fluidos, bem como alguns dos contextos em que estas foram utilizadas, e deste modo, selecionar a melhor abordagem para implementar estas técnicas e respetivos algoritmos, num motor de jogos como o Unity. Assim, em conformidade com o mais recente crescimento dos motores de jogo, pretendemos mostrar que estes motores podem contribuir para uma melhor implementação interativa de uma simulação de fluidos e, em simultâneo, manter o rigor científico das simulações. Com este propósito em mente, são apresentadas quatro simulações de fluidos, ou seja, uma simulação inicial, implementada com três otimizações de interação entre partículas, com o objetivo de testar qual a abordagem que melhor se enquadra no objetivo pretendido por um possível utilizador.

A solução proposta, foi implementada recorrendo ao método de computação da mecânica de fluxos de fluido CFD, *Smoothed-particle hydrodynamics*(SPH) com o propósito de simular todo o corpo do fluido, com interação entre partículas. Esta interação foi calculada apoiando-se em quatro algoritmos de procura de vizinhança diferentes, um algoritmo que computa a interação entre todas as partículas, um que recorre à simetria de forças pela terceira lei de Newton, outro inspirado pelo método de simulações de Monte-Carlo e por fim, um método híbrido em que se recorre a uma grelha auxiliar responsável por realizar o mapeamento das partículas.

Palavras-chave: Simulações 3D, Simulação de fluidos, Simulação de fluidos em Unity, Simulações baseadas em grelhas, Simulações baseadas em partículas, CFD, SPH .

Abstract

Fluid simulation is composed by a set of techniques and tools that aim to emulate the behavior of fluids in different scenarios. Nowadays, there is a wide variety of applications that use this type of simulations, which can be used for both scientific investigations and entertainment, such as movies and videogames. Simulations like these can represent various types of fluid, such as: rivers, oceans, winds, current flows, or even lava. As such, in order to create fluids, different aspects of their behavior must be considered, such as velocity, fluid density, fluid pressure or its viscosity level.

In this thesis, we intend to investigate fluid simulation techniques as well as some of the contexts in which they are used. This way, we pretend to select the best approach to implement these techniques and respective algorithms in a game engine such as Unity. In line with the latest growth in game engines, we intend to show that these can contribute to a better interactive implementation of a fluid simulation and, at the same time, maintain the scientific rigor of the simulations. With this purpose in mind, four simulations of fluids are presented. These are, an initial simulation, implemented with three optimizations of interaction between particles, with the objective of testing which approach best fits the objective intended by a possible user.

The proposed solution was implemented using the fluid flow mechanics computation method CFD, *Smoothed-particle hydrodynamics*(SPH) to simulate the entire fluid body with interaction between particles. This interaction was calculated using four different neighborhood search algorithms, one algorithm that computes the interaction between all particles, one that resorts to the symmetry of forces by Newton's third law, another inspired by the Monte-Carlo simulation method and finally, a hybrid method using an auxiliary grid responsible for mapping the particles.

Keywords: 3D Simulation, Fluid simulation, Unity fluid simulation, Grid based simulations, Particle based simulations, CFD, SPH.

Índice

Índice de Figuras

Índice de Listagens

Siglas

CFD Computational Fluid Dynamics (Dinâmica de Fluidos Computacional)

FAB Fluxed Animated Boundary (Animação de Fronteiras de Fluxos)
FFT Fast Fourier Transformation (Transformação rápida de Fourier)
FPS Frames per second (Quadros por Segundo

PBD Position Based Dynamics (Dinâmica Baseada na Posição)

SPH Smoothed particle hydrodynamics (Hidrodinâmica de Partículas Suavizadas)

1

Introdução

Existem diversos tipos de simulações do mundo real, por exemplo as simulações de fluidos, a formação de galáxias, ou até mesmo, simulações de Monte Carlo. Entre estas, uma das mais solicitadas é a de Simulação de Fluidos, que pode ser utilizada para as mais diversas finalidades. Ao longo dos anos, surgiram duas formas principais de interpretar esta categoria de simulações, a Animação de fluidos e a Dinâmica de Fluidos Computacional, CFD (Computational Fluid Dynamics) que podem acabar por ser confundidas, mas na realidade, uma é usada principalmente para efeitos visuais, enquanto a outra, serve, de facto, para estudar o comportamento dos fluidos de uma forma mais rigorosa e científica, respetivamente.

A crescente procura por estas simulações, provém do facto de, ao contrário do que se possa imaginar, a simulação de fluidos propõe muito mais aplicações para além dos efeitos visuais. De facto, antes mesmo do setor da computação gráfica começar a trabalhar nesta área, já se modelavam estes fluidos de uma forma ativa, com uma abordagem mais algébrica. Nos anos 50 e 60 um grupo de investigação liderado por Francis H. Harlow desenvolveu métodos como *particle-in-cell* [18], *fluid-in-cell* [16], *vorticity-stream function* [12] e *marker-and-cell* [19] sendo estes métodos uma visão numérica para simular fluxos de fluido transitórios e bidimensionais. Além disso, muito antes desta década, entre os anos de 1750 e 1850, já os físicos Euler, Navier e Stokes tinham chegado às conhecidas equações de Navier-Stokes, usadas para modelar de forma precisa a maioria dos fluxos de fluidos que ocorrem na natureza. Desde então, tem-se vindo a aprofundar cada vez mais a sua investigação, descobrindo-se mais funcionalidades para as equações por de trás destes fluidos, podendo-se descrever de uma forma física, problemas em áreas como a engenharia ou até mesmo modelar correntes de água no oceano, o comportamento da água em canos ou inclusivamente, o ar que passa pelas asas de um avião. Um bom exemplo que demostra esta importância e até mesmo necessidade, é um estudo realizado por uma equipa de investigadores [52], que consiste num modelo de interação entre glaciares e o próprio oceano, para a criação de tsunamis por parte da fragmentação de *icebergs*. Este estudo explora uma das muitas características que se pode querer representar num fluido, que é a interação de sólidos com o fluido em questão. Para isso, foi criado um método

com o objetivo de modelar uma fragmentação dinâmica do *iceberg* por parte de uma combinação da gravidade com a flutuação do mesmo, e ainda, a propagação da ondulação em formato de *tsunamis* produzida por esta fragmentação. Este artigo, para além de poder vir a dar grande contribuição para a comunidade científica, na recente questão das mudanças climáticas e ainda possibilita melhorar a avaliação de perigos para regiões costeiras e que medidas tomar para a sua mitigação [52].

Outro setor, que nos últimos anos tem vindo a recorrer à simulação de fluidos com elevada frequência é o do Entretenimento, onde os filmes e videojogos tentam alcançar ambientes digitais mais realistas e apelativos para os consumidores.

Inicialmente, de forma a reproduzir efeitos de água, estes tinham de ser desenhados em computadores, à mão e imagem por imagem, devido à complexidade que as equações para calcular as físicas de um fluido apresentam. Isto até 1998, em que AntZ[1] se torna um dos primeiros filmes de animação a utilizar fluidos computados por computadores, apesar da sua complexidade e exigência para a época[2]. Após esta iniciativa, em 1999 Jos Stam, [43] criou o que se vem a tornar a base para todo este mercado. O que Stam demonstrou neste estudo, foi uma forma mais simples das equações das dinâmicas de fluidos (fluid-dynamics equations) e que não exigem computadores tão potentes. Esta revolução levou a uma integração destas funções em *softwares* de animação, que contribuíram para o desenvolvimento dos efeitos visuais de grandes filmes, como foi o caso do Lord Of The Rings e do Pirates of the Caribbean. Contudo, esta abordagem apresentava algumas limitações em relação ao controlo que os animadores tinham sobre o fluido, e o tipo de fluido que era possível produzir, já que era impossível lidar com fluidos como a água[3]. Desde então, tem-se vindo a realizar várias investigações com o objetivo de melhorar estes resultados, e assim, proporcionar efeitos cada vez mais realistas, deixando os consumidores ainda mais impressionados.

De facto, recentemente, simulações de fluidos são cada vez mais usadas neste setor. Bons exemplos que o demonstram, foi a sua utilização para a representação da água e neve, respetivamente, nos filmes da Disney, Moana (2016)[4] e Frozen (2013) [5]. Dito isto, as aplicações de computação gráfica não precisam de gerar resultados fisicamente corretos (ao contrário das CFD) desde que estes pareçam convincentes e casualmente atraentes para o observador. Estas aplicações podem variar entre simulações em tempo real posteriormente integradas em videojogos e simulações ultrarrealistas, usadas principalmente em efeitos especiais na indústria cinematográfica.

[1] AntZ, imdb, Link:https://www.imdb.com/title/tt0120587/?ref_=nv_sr_srsg_0, último acesso: fev 2022

[2] Guinness World Records, Link:https://www.guinnessworldrecords.com/world-records/first-film-with-digital-water, último acesso: fev 2022

[3] Popular science, Link: https://www.popsci.com/entertainment-gaming/article/2008-02/and-oscar-goes-fluid-simulation-algorithms/, último acesso: fev 2022

[4] Moana, imdb, Link: https://www.imdb.com/title/tt3521164/?ref_=fn_al_tt_1, último acesso: fev 2022

[5] Frozen, imdb, Link: https://www.imdb.com/title/tt2294629/?ref_=fn_al_tt_1, último acesso: fev 2022

1.1 Motivação

Uma das razões pela qual os videojogos atraem tantas pessoas, é que tal como nos filmes, é possível criar mundos virtuais com as mais variadas características, sejam elas a aparência com o mundo real, ou algo completamente fictício. Os videojogos ainda têm a particularidade de oferecer ao utilizador a capacidade de interagir verdadeiramente com esses mundos e com aquilo que os rodeia. Sendo que, os fluidos surgem em praticamente toda a parte da natureza, como o oceano, o fumo, ou, até mesmo, uma pequena lareira, adicionar funcionalidades com a capacidade de reproduzir estes fenómenos, torna-se cada vez mais crucial para tornar os ambientes imersivos. Criar estes efeitos de fluidos, por si só, pode ser bastante desafiante e demorado, especialmente, para que sejam convincentes. Quando se fala de videojogos, para além destas características, ainda se deseja que seja possível interagir com o fluido em tempo real. Para isto acontecer, é necessário recorrer a algoritmos muito mais eficientes que os das simulações descritas anteriormente. É necessário que sejam capazes de criar os fluidos de forma convincente e ainda de forma rápida e em tempo real, tornando a sua tarefa muito mais complicada. Outro fator que contribui para a dificuldade de implementar a simulação de um fluido, fisicamente guiado, é que para atingir uma simulação rápida, a utilização de múltiplas *threads* é muitas vezes o recurso predileto, o que para um utilizador menos experiente em paralelização pode originar grandes problemas ao realizar uma adaptação da aplicação para uma arquitetura *multi-core*.

Todos os fatores acima referidos, geram um problema para um determinado grupo de utilizadores que pretenda criar para o seu jogo ou aplicação, uma simulação de fluidos fisicamente guiada, com interação e computada em tempo real, sem que para isso tenha de recorrer a paralelização. Considerando as necessidades e fatores anteriormente citados, nesta dissertação propõe-se, com recurso a um motor de jogo, a criação de um simulador de fluidos, com diversas otimizações e parametrizações, de forma a que seja possível adicionar a um jogo e ou aplicação, uma simulação de um fluido regido pelas CFD (Computational Fluid Dynamics) sem o recurso a paralelização, de forma a que seja admissível a interação em tempo real com outros objetos rígidos. O motivo para a utilização destes motores, deve-se ao seu mais recente crescimento tecnológico e ao elevado nível de acessibilidade, tendo-se tornado uma das principais ferramentas utilizadas para o desenvolvimento deste tipo de aplicações interativas.

1.2 Questões de Investigação

O foco desta dissertação foi investigar diferentes métodos de simulação de fluidos, em diversos contextos, e de seguida, definir, implementar e testar estes algoritmos e possíveis otimizações. Para guiar este trabalho de investigação, arquitetura e implementação do sistema que se propõe, foram definidas certas perguntas de investigação que são enumeradas e contextualizadas no seguimento da presente secção.

Como sugerido, para guiar a nossa investigação, foram definidas as seguintes questões:

Q1: Será possível criar uma simulação de fluidos interativa num motor de jogos, mantendo o comportamento fisicamente realista e esperado da simulação.

Q2: Como criar um sistema de simulação de fluidos parameterizáveis, numa cena tridimensional, que possibilite personalização.

Q3: Quais os melhores algoritmos de simulação de fluidos, gerados por interação de partículas, para reproduzir uma simulação eficiente, realista e adequada ao cenário/comportamento que se pretende simular.

A primeira questão, tem como finalidade esclarecer se um motor de jogos é capaz de comportar um simulador de fluidos e desta forma justificar a sua utilização, aplicabilidade, e como dispor de um sistema de simulação de fluidos pode ser uma mais valia para estas completas ferramentas de desenvolvimento.

A questão seguinte, remete para o modo como se pretende que o sistema implementado seja capaz de simular diversas classes de fluidos líquidos. Esta questão pretende esclarecer como o nosso sistema será implementado, para que o fluido e cena disponham de uma variada lista de parâmetros de especificação.

Para finalizar, a terceira e última questão apresentada, está relacionada com o modo como o algoritmo de simulação de fluidos está implementado, tendo à sua disposição diversas otimizações que devem ser analisadas, com o objetivo de comparar a sua eficiência e verificar quais os cenários em que cada uma das implementações pode demonstrar um melhor rendimento, em comparação com as demais alternativas.

1.3 Objetivos

Com o objetivo de responder às questões de investigação anteriormente estabelecidas, procedeu-se à implementação de um sistema de simulação de partículas em tempo real, guiadas por CFD com três otimizações dispares com base nas tão conhecidas equações de Navier-Stokes. Posto isto, considerando as questões de investigação anteriormente definidas, o desenvolvimento desta dissertação, foi dividido na seguinte lista de objetivos:

1. Investigação das bases e evolução do estado atual das simulações de fluidos guiadas por CFD e as suas contribuições em aplicações interativas, como é o caso dos vídeo jogos.

2. Definição de uma lista de funcionalidades e propriedades a serem implementadas, e distinção das diferentes otimizações aplicadas na computação do fluido.

3. Planeamento dos diferentes cenários de simulação, especialmente desenhados com a finalidade de colocar em prova as diferentes funcionalidades anteriormente estabelecidas.

4. Implementação de um simulador de fluidos com diferentes níveis de otimização, guiado por CFD, que cumpre com os diferentes requisitos delineados anteriormente, e que seja capaz de correr a simulação em tempo real, com interação e com diferentes variáveis totalmente parametrizáveis.

5. Avaliação do protótipo finalizado, através de diferentes estatísticas, com o objetivo de analisar a qualidade dos resultados e visualização dos diferentes cenários simulados, de forma a verificar os diferentes comportamentos suportados pelo fluido.

1.4 Contribuições

As principais contribuições alcançadas por esta dissertação, são a utilização de diferentes algoritmos de busca e interação de partículas, como forma de otimizar as computações necessárias para a criação de simulações de fluidos com o método de *Smoothed-particle hydrodynamics* (SPH). Estas otimizações permitem que o utilizador crie diferentes formas de simulação de fluidos, com certos níveis de velocidade computacional, sem recorrer à utilização de computação paralela. Com estas otimizações, é possível criar uma simulação apenas guiada pela interação independente das partículas, assim como, recorrer a uma grelha estática auxiliar na busca das partículas em interação. Em adição, também criámos diversas áreas de simulação para que seja possível observar os diferentes cenários/comportamentos suportados pelo nosso simulador. Assim, com uma lista de propriedades de fluido completamente personalizáveis, é possível criar diferentes comportamentos como a ondulação da água, fluxos em cascata, ou uma simples queda de um grande volume de fluido líquido, tal como pode ser observado no exemplo da Figura 1.1. Por fim, terminada a implementação do nosso simulador, foram realizadas diversas comparações estatísticas entre as diferentes otimizações disponíveis, sendo possível chegar a algumas conclusões em relação à aplicabilidade destas otimizações, tal como o grau de eficácia entre si. Para além das avaliações estatísticas anteriormente citadas, também foi importante a avaliação dos nossos fluidos quando colocados nos diferentes cenários de simulação.

1.5 Estrutura do Documento

Este capítulo inicial, introduz o contexto e a motivação para o desenvolvimento desta dissertação, onde é apresentada uma pequena introdução histórica da simulação de fluidos, assim como, a importância desta área de investigação e quais as oportunidades que origina nos diferentes setores. Por sua vez, é feita uma pequena descrição do problema, juntamente com as questões que este origina. Por fim, com base nas questões apresentadas, são propostos os objetivos a cumprir e as principais contribuições que se pretendem alcançar.

Segue-se o segundo capítulo, onde são estabelecidas algumas noções básicas da área e a apresentação de alguns trabalhos relacionados. Começa-se pela apresentação das

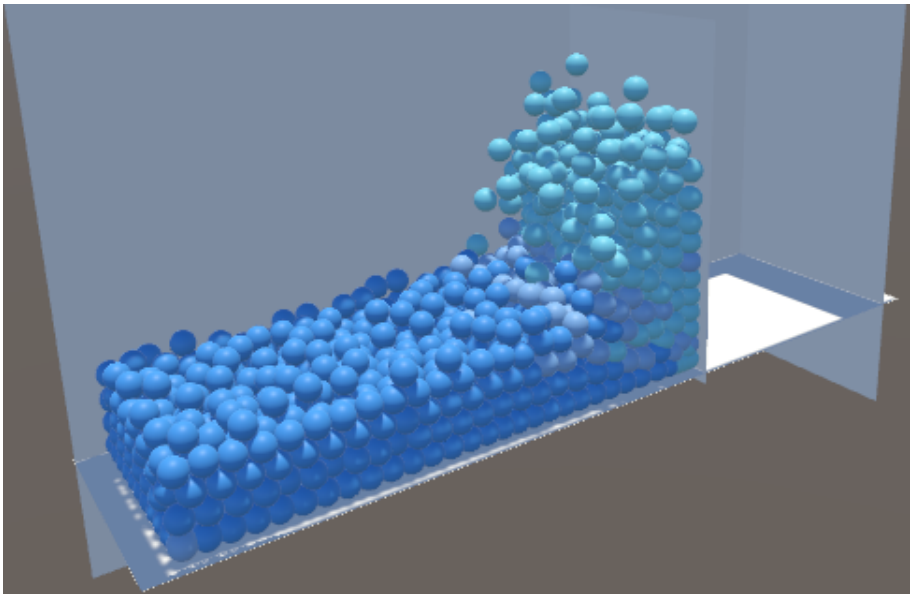


Figura 1.1: Exemplo do comportamento de ondulação do fluido, quando condicionado pela ação de um obstáculo em movimento.

equações de Navier-Stokes, Boundary Conditions, seguido-se da explicação dos principais pontos de vista no que toca às simulações. Passa-se então, para as simulações em tempo real e termina-se numa secção para a Interação e Computação gráfica, onde será descrito o funcionamento geral de um motor de jogos e alguns videojogos.

O terceiro capítulo, começa com uma descrição mais detalhada do sistema que se propõe neste estudo, onde são apresentados os requisitos, a arquitetura, as principais ferramentas, e por último, é explicada em maior detalhe, a teoria utilizada como base de desenvolvimento do sistema implementado. No quarto capítulo, é descrito o funcionamento geral do sistema, juntamente com os atributos necessários para a sua utilização. De seguida, é apresentada a implementação e o funcionamento de cada um dos algoritmos implementados, seguindo-se uma secção onde são descritos os diversos cenários e modos de observação que o nosso sistema de simulação de fluidos pode oferecer.

O quinto capítulo, começa por explicar as diferentes métricas de avaliação selecionadas para testar o funcionamento de cada um dos algoritmos e cenários. Deste modo, é realizada a avaliação e descrição do comportamento e os resultados observados no teste das referidas métricas. Adicionalmente, segue-se uma secção final, onde é possível observar os diferentes fluidos em interação com os diferentes cenários, resultando assim, nos comportamentos suportados pelos fluidos do nosso sistema. Para finalizar, no sexto capítulo, é apresentada a conclusão que se retira de todos os testes e cenários implementados, e consequentemente, as perguntas de investigação propostas serão respondidas com base nos resultados obtidos.

2

Enquadramento e Trabalho Relacionado

O objetivo que se pretende atingir com este capítulo é fornecer uma visão geral em relação as áreas de estudo relacionadas com simulação de fluidos. Assim, será feita uma pequena análise histórica de determinados trabalhos relevantes para o objetivo da presente dissertação, que em combinação com fundamentos teóricos referentes ao comportamento de fluidos, poderão servir como base para o desenvolvimento se um simulador de fluidos. Em primeiro lugar, será feita uma breve passagem pela definição e história das simulações de fluidos e que questões estas acatam. Seguidamente, serão analisados os principais conceitos como base para a computação de fluidos. Com todas a noções anteriores enunciadas, serão apresentadas diversas técnicas de implementação de simulações de fuidos juntamente com os requisitos destes sistemas. Para finalizar, será feita uma breve explicação do que consiste um motor de jogos e serão analisadas algumas implementações de fluidos em dois dos principais video jogos do mercado. Será ainda explicada, de que forma estas simulações causam tanto impacto na referida indústria.

2.1 Técnicas de Simulação de Fluidos

Existem fluxos de fluido por praticamente todo o lugar que se possa imaginar, podendo estes ser o simples ar que se respira, como os rios e oceanos que nos rodeiam. Para além destes impressionantes fenómenos e da sua interação com objetos rígidos, inclusivamente com seres vivos, podem ser gerados pequenos salpicos de água e até mesmo gases, como o fumo de uma fogueira e inclusive, as próprias chamas. Deste modo, pelo fato de todos estes fenómenos estarem presentes no dia a dia, rapidamente os fluidos se tornaram uma importante parte na computação gráfica.

Esta importância, advém da necessidade de através da simulação de fluidos se ter uma melhor perceção de como estes funcionam e assim, se desenvolverem novas tecnologias com base nesse conhecimento. Uma área para a qual a simulação de fluidos também tem ganho bastante importância é a indústria de desenvolvimento de videojogos, visto que

um dos seus principais objetivos é criar mundos cada vez mais imersivos para os jogadores. Originando assim, a demanda por essas tecnologias para motores de jogos, como é o caso do Unity e até mesmo do Unreal. Com este objetivo em mente, Jos Stam [42] afirma que, apesar de já existirem vários modelos que têm a finalidade de simular efeitos que se assemelhem com fluidos, apresentando ainda um método que consistia em renderizar partículas como texturas, estes modelos apresentam um problema que consiste na dificuldade que existe em animar estas mesmas partículas de forma convincente.

Posto isto, acredita-se que a melhor alternativa para resolver este problema, seja o recurso a abordagens com base na física dos fluxos de fluidos. Estas abordagens, que começaram a ser desenvolvidas desde 1750, originaram as incompressíveis equações de Navier-Stokes que continuam lideres no que toca à simulação de fluidos.

2.1.1 Termos usados pelas Equações de Navier-Stokes

Advecção: A movimentação do fluido faz com que o mesmo transporte objetos e outros fluidos, seguindo o seu fluxo. Imagine-se que, ao esguichar corante para um fluido em movimento, o corante é transportado ou adveccionado, ao longo do campo de velocidade do fluido.

Nas Navier-Stokes, o termo de Advecção transporta a velocidade de um ponto para outro no seu campo de velocidade. Isto significa que esta é responsável por ajudar a evolução do campo de velocidade, no decorrer do tempo.

Pressão: As moléculas de um fluido podem mover-se em volta umas das outras e tendem a espremer e a comprimir-se. Isto acontece quando uma força é aplicada ao fluido, apesar de não percorrer o fluido completo de forma instantânea. Ao invés disso, as moléculas que se encontram mais próximas do ponto de aplicação da força empurram as que se encontram mais distantes e, devido a este fenómeno, a pressão aumenta. Como a pressão é a força por unidade de área, qualquer pressão no fluido naturalmente leva à sua aceleração.

Difusão: Ao interagirmos com os fluidos do mundo real, é percetível que nem todos têm a mesma espessura. Perante este facto, é costume dizer-se que fluidos espessos possuem maior viscosidade. A viscosidade é a medida de quanto um fluido resiste para fluir. Essa resistência resulta na difusão do momento (e, portanto, da velocidade), e por isso, é chamada difusão. Por outras palavras, podemos ver esta força, como a força responsável por fazer uma partícula do fluido mover-se na velocidade média das suas partículas vizinhas.

Forças externas: Por fim, temos a forças externas aplicadas ao fluido. Estas forças podem ser forças locais ou Forças de corpo (*Body forces*). As forças locais são aplicadas a uma região especifica do fluido, por exemplo, a força de uma ventoinha a soprar o ar. Por outro lado, as Forças de corpo, são aplicadas de forma constante por todo o fluido. Um bom exemplo de uma força desta classe é a própria gravidade.

2.1.2 Navier-Stokes

Como dito anteriormente no capitulo 2.1, as equações incompressíveis de Navier-Stokes encontram-se atualmente como o epicentro da maioria dos problemas que provêm das *Computational Fluid Dynamics* (CFD), um conjunto de equações diferenciais parciais que são encarregues de se manterem em todo o fluido, e que controlam o movimento dos fluidos reais e com viscosidade.

$$\frac{\delta \vec{u}}{\delta t} + \vec{u} \cdot \nabla \cdot \vec{u} + \frac{1}{\rho} \nabla p = \vec{g} + v \nabla \cdot \nabla \vec{u} \tag{2.1}$$

$$\nabla \cdot \vec{u} = 0 \tag{2.2}$$

- A variável $\vec{u}$ é tradicionalmente usada em mecânica de fluidos que corresponde à velocidade do fluido.

- A variável dada por ρ representa a densidade do fluido, onde para a água, esta é aproximadamente 1000kg/m3 e para o ar, esta já toma o valor de 1.3kg/m3, correspondendo a uma proporção de 700:1.

- A variável p representa a pressão, que corresponde à força que é aplicada por unidade de área que o fluido exerce em qualquer coisa.

- A variável $\vec{g}$ é a já conhecida aceleração da gravidade (0,-9.81,0)m/s2, em combinação com as restantes forças externas, que também atuam sobre as partículas.

Algo importante de notar, é que por convenção o eixo do y aponta verticalmente para cima e os eixos do x e z equivalem aos eixos horizontais.

Isto deve ser adicionado em animações como forma de garantir um controlo adicional da aceleração (com o propósito de fazer o fluido comportar-se da forma desejada) que pode ser adicionada sobre a própria gravidade. Todas estas forças originarão então, o valor de $\vec{g}$. Estas forças costumam ser conhecidas pelo nome de Forças de corpo (*Body forces*), já que são aplicadas em todo o corpo e não apenas numa certa área.

- Por fim, a variável v é tecnicamente chamada de viscosidade cinemática *Kinematic viscosity*. Esta é definida pela proporção da viscosidade dinâmica μ e pela densidade do fluido ρ, que mede o quão viscoso é o fluido. Fluidos como o mel têm um alto nível de viscosidade, enquanto que outros como o álcool, apresentam um nível mais baixo. Isto é o que quantifica a capacidade de resistência do fluido a deformações enquanto este flui.

2.1.3 The Momentum Equation

A primeira equação, normalmente conhecida pelo nome *The Momentum equation*, na realidade pode ser vista como uma junção de três equações mais simples que correspondem a diferentes forças, podendo de facto representar a aceleração adquirida por parte do fluido por ação das forças que lhe atuam.

De forma a explicar como esta equação funciona, Robert Bridson [7] afirma que o objetivo é de simular um fluido recorrendo a um sistema de partículas onde cada partícula

deve representar um *blob* de fluido. Para isso, deve conter uma massa m, volume V e velocidade dada pelo $\vec{u}$. Posteriormente, para integrar o sistema no decorrer do tempo, tudo o que é necessário para descobrir que forças atuam em cada partícula são: $F = ma$ que indica de que forma estas aceleram, sendo assim possível, obter o seu movimento.

$$\vec{a} \equiv \frac{\delta \vec{u}}{\delta t} \tag{2.3}$$

$$m\frac{\delta \vec{u}}{\delta t} = \vec{F} \tag{2.4}$$

Ao iniciar a decomposição desta equação, rápidamente se repara que das forças que atuam nas partículas, a mais simples, é de facto a já conhecida gravidade, dada por: mg, sendo m a massa da partícula e g a aceleração da gravidade.

Mas a pergunta que pode ficar é, o que realmente torna estas partículas num fluido? A resposta de Robert Bridson é que efetivamente o restante fluido também exerce forças e que estas também devem ser consideradas. Dito isto, por outras palavras, significa que todas as outras partículas interagem também com as restantes que se encontram próximas a ela.

Força da Pressão: Para isto ocorrer, a primeira força a ser considerada é a chamada Pressão, que é responsável por manter um dado fluido t com o seu volume constante. Algo importante de notar, é que regiões de alta pressão empurram as de pressão mais reduzida. Dito isto, o que é realmente relevante de analisar para chegar a estas forças, é a força resultante (*net force*: soma de todas as forças) e.g "Se a pressão for igual em todas as direções então a força resultante será de zero e por isso não existirá aceleração causada pela pressão".

Para concluir, tal como é dito por Robert Bridson [7] o modo mais simples de medir o desequilíbrio da pressão na posição de uma dada partícula é simplesmente tomá-lo pela negação do gradiente da pressão: $-\nabla p$ e de seguida integrá-lo sobre o volume do *blob* do fluido de maneira a chegar à força da pressão.

Força da Viscosidade: Por fim, a restante força a atuar no fluido trata-se da Viscosidade, responsável por medir de que forma o fluido tenta resistir a deformações. A forma que Robert Bridson tenta explicar esta força, para que seja mais fácil de entender, é que se trata de uma força que tem como objetivo fazer com que uma dada partícula se mova à velocidade média das restantes partículas próximas desta. Distinguindo ainda, que o operador diferencial responsável por medir o quão longe uma certa quantidade se encontra da média em seu redor é o operador de Laplace: $\nabla \cdot \nabla$.

Disto, resultará a força de viscosidade do fluido, que, de seguida, após ser integrada sobre o volume do *blob*, esta força será completada com o coeficiente de viscosidade dinâmica, normalmente dado pela variável μ.

De notar, que tal como é dito por Robert Bridson, isto aplica-se apenas para fluidos em que a viscosidade não varie, pois nesse caso, este termo seria muito mais complicado.

Finalmente, unindo todas estas forças e após algumas simplificações, é chegada a equação final:

$$\frac{D\vec{u}}{Dt} + \frac{1}{\rho}\nabla p = \vec{g} + v\nabla \cdot \nabla\vec{u} \tag{2.5}$$

2.1.4 Incompressible Equations

É sabido que fluidos reais, até mesmo líquidos como a água, podem ter um volume que varie. Esta afirmação pode de facto parecer errada, mas como dito por Robert Bridson, pode-se muitas vezes pensar que a diferença entre líquidos e gases é que os gases podem mudar de volume, mas que de facto os líquidos não. Obviamente isto não está correto, já que se assim fosse, seria impossível ouvir sons de baixo de água.

No entanto, o que importa é que normalmente os fluidos não mudam de volume com grande frequência, por conta da quantidade de energia necessária para lidar com muitas mudanças no volume do líquido.

O estudo de como os fluidos se comportam nestas situações é normalmente chamado de fluidos compressíveis. Simular pequenas perturbações num volume tende a ser bastante caro e complicado, o que em combinação com o facto de essas pequenas perturbações constituírem um impacto tão reduzido no movimento do fluido a um nível macroscópico, estas ficam praticamente irrelevantes de animar. Isto significa que numa animação é possível tratar todos os fluidos como sendo incompressíveis, significando que o seu volume não varia.

Uma forma rápida de observar como é que isto é calculado (explicação mais complexa [7]) é selecionando um pedaço *Chunk* arbitrário do fluido e observá-lo num determinado instante de tempo. De seguida, é possível verificar o quão rápido o volume deste pedaço do fluido está a variar ao integrar a componente normal da sua velocidade em volta dos seus limites.

$$\frac{d}{dt}Volume(\Omega) = \int\int_{\partial\Omega} \vec{u}\cdot\hat{n} \tag{2.6}$$

Para um fluido incompressível, o volume deve manter-se constante, em função disso, a taxa de variação deve ser zero:

$$\int\int_{\partial\Omega} \vec{u}\cdot\hat{n} = 0 \tag{2.7}$$

De seguida, é possível usar-se o teorema da divergência para alterar isto para o volume integral:

$$\int\int\int_{\Omega} \nabla\cdot\vec{u} = 0 \tag{2.8}$$

E finalmente, para qualquer valor de Ω (qualquer região do fluido) esta equação deve manter-se verdadeira. A única função que se integra sempre a zero independentemente do volume de integração, é o próprio zero. Assim, o integrando deve ser sempre zero:

$$\nabla \cdot \vec{u} = 0 \tag{2.9}$$

2.1.5 Boundary Conditions

Até ao momento, esta dissertação tem estado mais assente naquilo que ocorre no interior dos fluidos, mas algo igualmente importante e que também faz parte de uma simulação de fluidos, são as suas fronteiras. Estando-se perante a simulação de fluidos, estes podem interagir com o recipiente em que se encontram, com objetos que estejam embutidos no fluido, ou até mesmo com outros fluidos que não se misturem (Como é o exemplo do óleo com a água). A estas interações que podem ocorrer é dado o nome de *Boundary Conditions*. De outra forma, descreve-se as *Dynamics equations* como a família dos movimentos, todas as trajetórias de todos os projeteis enquanto as *Boundary Conditions* representam o formato do recipiente que aguenta o fluido. De salientar, que cada umas das equações contém as suas próprias *Boundary Conditions* isto é, aplica-se as equações, de Momentum, de pressão e da densidade.

Os dois tipos de *Boundary Conditions* mais usados são, *Solid Walls* e *Free Surfaces*. Uma *Solid Wall boundary* é quando o fluido entra em contacto com um sólido, que analisado pelo ponto de vista do termo da velocidade, o fluido não pode entrar no sólido nem sair dele. Para que haja esta garantia, a componente normal da velocidade tem de ser zero, caso o sólido não se esteja a mover, contudo, se isto não for verdade (ou seja o sólido estiver em movimento) a componente normal da velocidade do fluido tem de ser igual à do sólido em questão. Portanto, é importante clarificar a velocidade relativa do sólido e do fluido, podendo ainda ter-se em consideração as influências térmicas que o fluido pode aplicar no sólido.

Uma *Free surface boundary*, é o termo utilizado para se referir a parte do *Fluid Boundary* que não está em contacto com paredes ou outros sólidos. Por exemplo, se estivermos a simular a água e os seus salpicos, as superfícies de água que não estão em contacto com nenhuma parede sólida são *Free Surfaces*. Neste exemplo, está mesmo a ser considerada a interação com outro fluido, o ar, que muitas vezes pode ser ignorado, já que a sua pressão é aproximadamente 700 vezes menor que a da água e assim basta atribuir à pressão do ar, uma contante igual a zero.

Outra situação em que este tipo de *boundary* pode ser útil, é quando se pretende simular um pouco de um fluido que faz parte de um com muito maiores dimensões (Simular fumo no ar livre). Numa situação como esta não é ideal simular toda a atmosfera, e por esta razão, a tendência é de criar uma grelha (*grid*) que contenha apenas as regiões que se espera que tenha algum interesse. Tendo isto em consideração, apenas essa região

é de facto simulada e para além dos limites dessa região, o fluido simplesmente continua sem uma simulação envolvida.

Na Figura 2.1 é possível ver os vários tipos de *Boundary Condition*. O que acontece

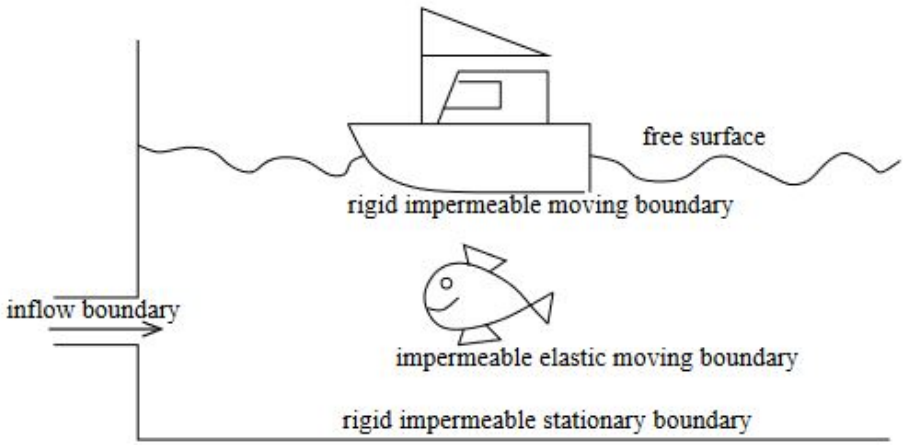


Figura 2.1: Imagem que mostra um exemplo da representação de cada *Boundary Condition* [1].

quando recorremos a um Simulador de fluidos baseado em físicas, é que enquanto é garantido que o input é feito de físicas, é possível garantir que será dado como retorno, um output fisicamente realista. O problema que pode surgir é que em muitas situações o Diretor exige um resultado que não seja fisicamente preciso, mas que contenha um aspeto mais "mágico/magnífico"como o que está apresentado na Figura 2.2.

Figura 2.2: Exemplo de uma forma artística usando FAB [44].

Com o objetivo de manipular o controlo necessário para chegar a esses resultados Alexey Stomakhin distingue três métodos, *Velocity overrides*, *Volumetric forces* e *Boundary Conditions*, cada um destes menos agressivo que os outros (pela ordem apresentada). Quanto mais agressivo é o método escolhido, mais controlo se terá do fluido, o que faz com que se perca um pouco a essência de ter uma simulação.

[1] https://personalpages.manchester.ac.uk/staff/matthias.heil/Lectures/Fluids/Material_2019/Chapter4.pdf

No estudo de Alexey Stomakhin et al. [44] o problema que se está a resolver é que no desenvolvimento do filme Moana [2], era preciso representar vários barcos a navegar no oceano, criar a simulação do contacto dos barcos com a água e fazer com que essa simulação combine com o resto do oceano, sem que fosse necessário simular o oceano por completo.

Para um problema mais geral, se for apresentado um cenário cheio de água, onde exista um corpo a viajar por esse cenário, é necessária uma simulação local apenas em redor do corpo. Após isso, ao correr a cena com esse corpo, é preciso impor uma certa *Boundary condition* de forma que essa simulação interna combine com o resto do fluido existente. Para resolver esse problema, como enunciado por Stomakhin et al. [44], o método mais intuitivo seria criar um objeto físico em volta do barco, este objeto acompanhava o barco enquanto este se movimenta, e a água no interior dessa área reagia ao movimento do barco (ver Figura 2.3). O problema nesta abordagem surge quando o barco se movimenta, porque assim, o objeto em seu redor acompanhava esse movimento, movendo assim a água contida no seu interior, ainda afetando a movimentação da água ao seu redor. Como o objetivo é que a água seja estática, então invés de se considerar um objeto físico, utiliza-se a chamada *Fluxed Animated Boundary* (FAB) separando assim o movimento da janela em volta do barco e uma segunda entidade que seria a água estática *procedural*, que se encontra no exterior da janela (ver Figura 2.4).

De notar, que esta abordagem já existia quando métodos de Euler eram mais comuns, mas o principal foco neste estudo foi o de integrar estas técnicas em simulações baseadas em partículas (Lagrange) utilizando *solvers* como o FLIP/APIC, possibilitando assim uma forma simples de apagar e criar partículas, o que torna a vida de um artista significativamente mais simples.

2.1.6 Algoritmo de Simulação e Funcionamento

A figura 2.5 representa uma abstração do funcionamento de um sistema de partículas ou nós de grelha do fluido. Após ser recebido o estado inicial, input do utilizador e o estado geral com todas as partículas, as forças, consideradas em 2.1.3 e 2.1.4, são calculadas tendo sempre em consideração, as *boundary conditions* (como explicado em 2.1.5) que lhe são impostas. Seguidamente, com a aplicação dessas forças, é atualizada a nova posição da particula, procedendo-se ao *render* onde serão aplicadas essas alterações. Após isso, volta-se ao cálculo das forças, considerando agora estas novas posições e novos inputs de utilizador (caso existam).

2.2 Pontos de Vista de Simulação

Após vários anos de investigação, já existem muitos métodos usados em *Computational Fluid Dynamics* (CFD) para simular fluidos. E tal como dito por Brakey et al. [6] entre as

[2]Moana, imdb, Link: https://www.imdb.com/title/tt3521164/, último acesso: fev 2022

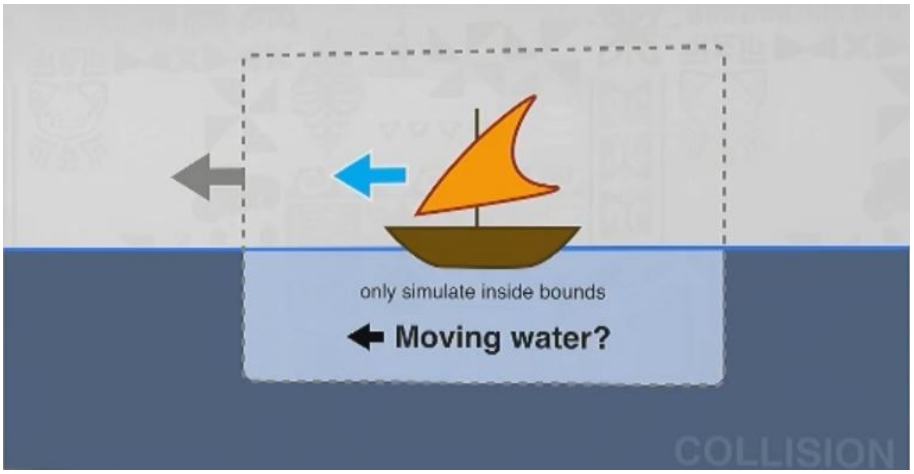


Figura 2.3: Interpretação da janela, usando objeto fisico [44].

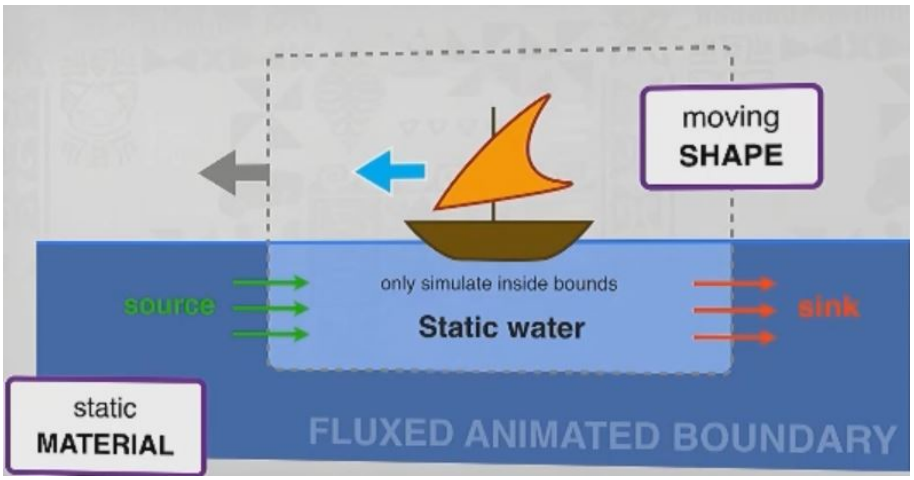


Figura 2.4: Interpretação da janela, recorrendo ao método FAB [44].

técnicas mais comuns encontra-se simulações baseadas em redes (*grid based simulations*) e simulações baseadas num sistema de partículas (*particle based simulations*). Sendo estas duas técnicas bastante distintas uma da outra, tipicamente simulações baseadas em redes funcionam com um nível de precisão bastante elevado resultando num processamento relativamente lento, que em comparação com simulações baseadas em partículas, apesar de serem bastante mais rápidas, tendem a ter uma aparência de qualidade inferior.

Quando se pretende representar algo continuum, como um fluido ou sólido deformável que esteja em continuo movimento, Robert Bridson e Matthias Müller-Fische em [3] distinguem duas abordagens para acompanhar o movimento/Acão do fluido, que dão pelo nome de *Lagrangian viewpoint* e *Eulerian viewpoint*, derivadas por dois importantes matemáticos. Estas abordagens são vistas como de grande importância, sendo que até aos dias de hoje, os vários métodos de *Computational Fluid Dynamics* (CFD) são classificados como sendo métodos de Euler, Lagrange ou até mesmo uma abordagem hibrida entre estas duas.

[3] Nome: SIGGRAPH 2007 Course Notes, Link: https://www.cs.ubc.ca/~rbridson/fluidsimulation/fluids_notes.pdf, último acesso: fev 2022

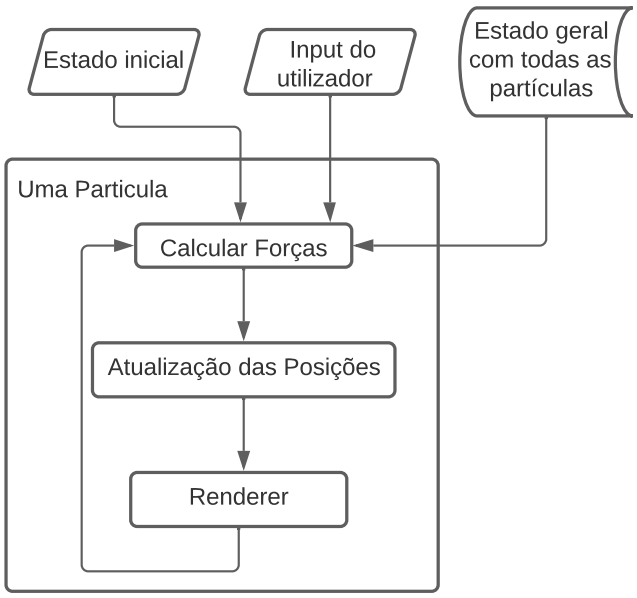


Figura 2.5: Representação do algoritmo de simulação.

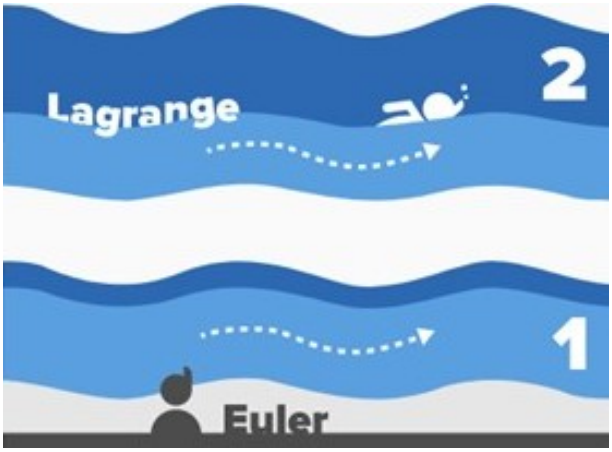


Figura 2.6: Imagem que descreve o exemplo do artigo [4].

De maneira a distinguir estas duas abordagens Robert Bridson's [7] apresenta: "Uma maneira de pensar nestes dois pontos de vista é ao fazer uma previsão meteorológica. Na visão de Lagrange é equivalente a estar num balão a voar ao sabor do vento, enquanto se mede a pressão, temperatura e humidade do ar que acompanha o movimento do balão. Por outro lado, a visão de Euler equivale a estar parado no solo enquanto se mede esses mesmos fatores do ar que vai passando."

Para se ter uma melhor impressão e mais relacionada com o objetivo desta dissertação, de como estes pontos de vista são interpretados, o artigo [5] apresenta um bom exemplo, com um rio com a água em movimento, onde o objetivo de Euler seria de observar o

[4]Nome: Dive, Link: https://www.dive-solutions.de/articles/cfd-methods, último acesso: fev 2022
[5]Nome: Dive, Link: https://www.dive-solutions.de/articles/cfd-methods, último acesso: fev 2022

fluir do rio enquanto se encontrava estático num único ponto de observação. Já Lagrange pretende observar este mesmo rio após fazer um mergulho e de seguida observar ao seu redor, enquanto se deixava levar pela corrente, como pode ser visto na Figura 2.6.

2.2.1 Visão de Lagrange - Modo de Sistema de Partículas

A abordagem de Lagrange (Joseph-Louis Lagrange) trata um continuum como um simples sistema de partículas. Podendo ser aplicado para fluidos ou até mesmo para sólidos deformáveis onde cada ponto nesse continuum é etiquetado como uma partícula individual, que contém uma posição $\vec{x}$ e velocidade $\vec{u}$, podendo cada uma ser imaginada como uma molécula constituinte do fluido [7]. Robert Bridson adiciona ainda que, no caso de sólidos, estes são quase sempre simulados pela forma de Lagrange, com um conjunto discreto de partículas que podem ou não estar conectadas como uma malha.

Um fluido pode então ser imaginado como uma vasta coleção de partículas em movimento, que depois de serem etiquetadas e lhes tenham sido atribuídas as suas várias propriedades, estas são observadas enquanto se movem pelo espaço tempo [27, 4].

Nas Figuras 2.7 e 2.8 está representada uma forma gráfica da visão de Lagrange.

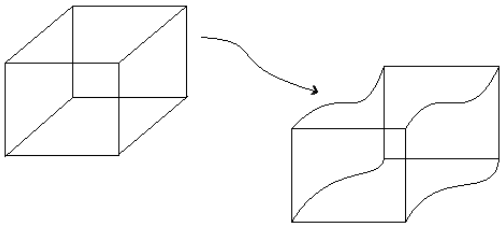


Figura 2.7: Visão de Lagrange: Observa-se cada ponto do fluido enquanto este acompanha o fluxo do fluido e a variação das suas propriedades.

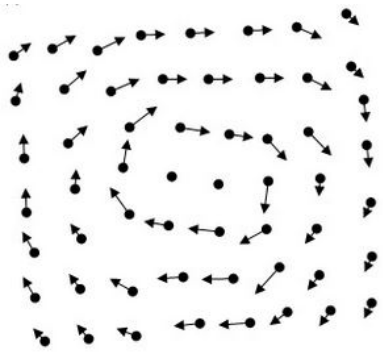


Figura 2.8: Representação do sistema de partículas [6].

2.2.2 Visão de Euler - Modo Grelha

A abordagem de Euler (Leonhard Euler) apresenta uma perspetiva completamente diferente, tendendo a ser mais utilizada apenas para fluidos, onde, ao invés de acompanhar o percurso de cada partícula, observa-se pontos fixos no espaço. Nota-se como as medidas das várias propriedades como a densidade, velocidade, temperatura e restantes características variam ao longo do tempo nesses pontos em específico. O fluido irá então atravessar através desses pontos, contribuindo assim para essa variação [7]. Tal acontecimento pode ser observado num exemplo dado por Robert Bridson, que consiste em observar que, enquanto um fluido mais quente se move através de um ponto específico, seguido por outro mais frio, a temperatura desse ponto no espaço irá diminuir apesar de

[6] Nome: venturebeat, Link: Sistema de partículas, último acesso: fev 2022

nenhuma partícula especifica do fluido estar a mudar. De fato, apenas é atravessado por algum fluido em movimento.

Nas Figuras 2.9 e 2.10 está representada uma forma gráfica da visão de Euler.

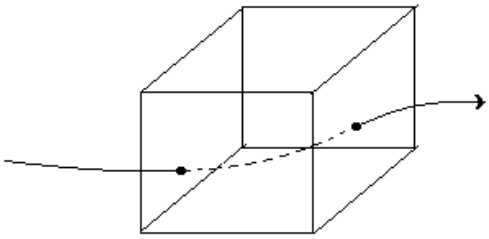


Figura 2.9: Visão de Euler: Ficar num ponto fixo no espaço e observar o fluido mover-se através do volume.

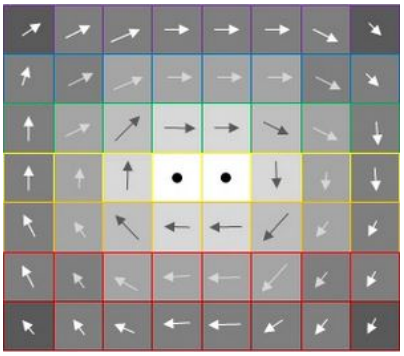


Figura 2.10: Representação do sistema em grelha [7].

2.2.3 Simulações Baseadas em Partículas

Nesta secção será feita uma descrição mais detalhada das abordagens baseadas em sistemas de partículas de Lagrange.

Para abordagens como esta, onde se recorre a técnicas descritas por Lagrange, estruturas baseadas em rede para simular os fluidos são colocadas de lado e passa a ser utilizado um conjunto discreto de partículas que se encontram em movimento no espaço. Ao optar por uma abordagem destas, como em qualquer decisão, tem de se estar disposto a fazer alguns sacrifícios de forma a obter algumas vantagens. O que acontece é que em abordagens deste tipo, a precisão acaba por ser inferior por conta das dificuldades que são impostas ao trabalhar com derivadas espaciais de uma nuvem de partículas que não esteja estruturada. Em contrapartida, simulações baseadas em partículas têm tendência a ser bastante mais simples de programar e muito mais rápidas de executar, sendo assim preferíveis para aplicações em tempo real como: vídeojogos [6], simuladores médicos ou qualquer tipo de ambiente virtual.

No que diz respeito à computação gráfica, desde algumas décadas atrás, se têm vindo a desenvolver cada vez mais técnicas para a simulação de fluidos:

Modelação de Objetos Difusos: Entre estas, William T.Reeves [39] introduziu a utilização destes sistemas de partículas, como uma técnica para modelar uma classe de objetos difusos como é o caso do fogo, nuvens e/ou água, onde estes sistemas moldam um objeto como uma nuvem de partículas primitivas, que assim definem o seu volume e, durante um certo período de tempo, essas mesmas partículas são criadas, movidas, alteradas e até mesmo destruídas nesse sistema.

[7] Nome: venturebeat, Link: Sistema com grelhas, último acesso: fev 2022

Neste modelo, através desses resultados, é então possível fazer a representação do movimento, dinâmicas e mudanças de forma, coisa que para representações em que apenas se baseava na superfície não era possível. Isto resulta no que se pode observar nas Figuras 2.11 e 2.12.

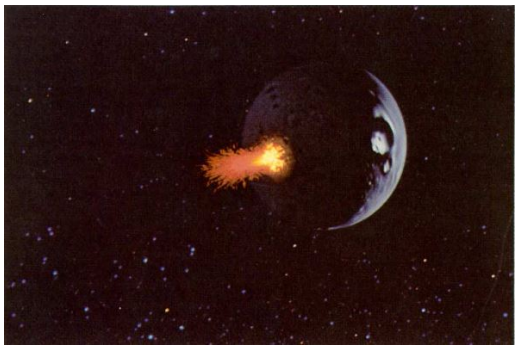


Figura 2.11: Explosão inicial [39].

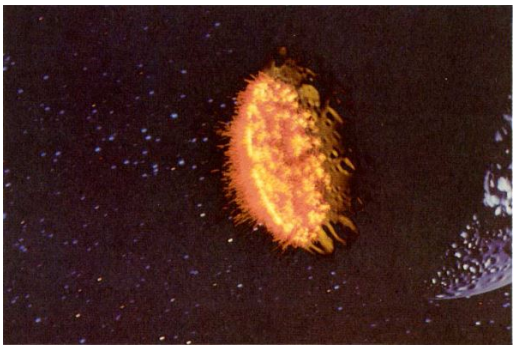


Figura 2.12: Propagação das chamas [39].

Animação de Objetos Altamente Deformáveis: Logo após isso, começaram a ser usados sistemas baseados em partículas com muito mais frequência e para os mais variados objetivos, entre eles, Mathieu Desbrun e Marie-Paule Cani et al. [11] apresentaram um novo formalismo para simulação de corpos altamente deformáveis (soft bodies) com sistemas de partículas, como representado na Figura 2.13. Para isso, recorrem a uma extensão do já conhecido modelo de Hidrodinâmica de partículas suaves (SPH - *Smoothed particle hydrodynamics*), uma abordagem que será descrita mais adiante no capítulo 2.2.4, mas que de forma simplificada foi criado e usado por físicos para simulação de fluidos cósmicos. Porém, considerando que, ao invés de ver as partículas como simples pontos de massa que descrevem um objeto, estes definiram as partículas suaves como amostras de massa perdidas pelo espaço. O que se concluiu neste estudo foi que quando estamos perante objetos modelados de forma suave, (*Smoothed*) continuam a ser fornecidas equações de movimento semelhantes às das partículas convencionais (forças aplicadas a um par, no centro de massa e simétricas). Para além disso, ainda desbloqueia muitas oportunidades para explorar.

A título de exemplo, a integração do tempo pode ser tratada de forma bastante rica, visto que cada partícula é integrada em momentos de tempo individuais, onde a escolha do momento de tempo atual é feita de acordo com diferentes critérios, de forma a assegurar a sua estabilidade.

Outras vantagens que se podem destacar, é que para além do que se referenciou acima, esta técnica também inclui, uma representação implícita coerente com o modelo físico derivado da densidade espacial, complexidade eficiente, e parâmetros fornecidos para o usuário escolher o tipo de material viscoso que se pretende animar.

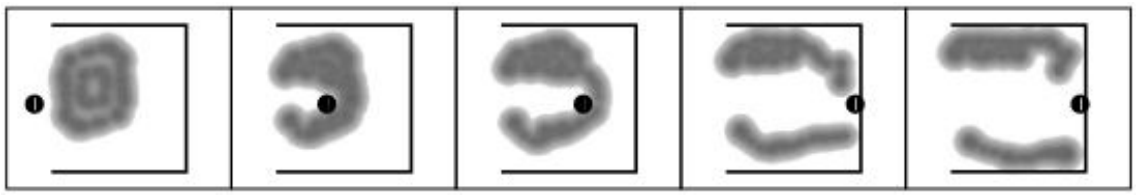


Figura 2.13: Animação em 2D, onde a densidade pode ser representada pelo nível de cinzento, que ilustra uma ferramenta rígida a curtar um objeto deformável em duas partes [11].

Modelação de Objetos com Memória: Outro exemplo da utilização de partículas foi apresentado em [30] para animar objetos com memória de forma. O que é proposto é uma nova abordagem baseada num sistema de partículas orientadas (*oriented particles*) por Szeliski et al. [46] para modelar e animar objetos com memória de forma, onde é necessária uma nova força de atração/repulsão chamada de força de coesão (*cohesion force*). Esta força é normalmente responsável por oferecer um melhor controlo sobre as oscilações dos sistemas de partículas, para além disso, de forma a impor restrições no formato do objeto, ainda é necessário descrever as formas de interação (chamadas de interações de forma). Como resultado, as partículas passam a poder ser usadas para simular superfícies 3D de forma livre e deformável, com a capacidade de retornar à sua forma original depois de sofrer a deformação, como é o caso de representar uma mola.

Controlo de Superfícies Implícitas: De forma a controlar superfícies implícitas [51], é apresentada uma nova abordagem que consiste em aplicar uma simples restrição, que prende um conjunto de partículas a uma superfície enquanto estas se movem em conjunto com a mesma, originando uma relação de dependência entre a superfície e essas partículas. De seguida, através da implementação de pontos de controlo é feita uma manipulação direta ao movimento das partículas, e, por sua vez, é resolvido o movimento da superfície mantendo a sua restrição com o objetivo de alcançar renderização em tempo real e manipulação direta de superfícies, de forma mais eficiente.

Para além disso, esta aplicação de métodos restritivos, ao aplicar repulsão local com o auxílio dos métodos de "Fissão" propostos, variando de forma adaptativa o raio de repulsão, é possível obter uma boa distribuição das amostras muito mais rapidamente do que os restantes esquemas de repulsão, mesmo estando sujeitos a rápidas mudanças de forma da superfície possibilitando assim, que a alteração das superfícies possa ser muito mais rápida.

Modelação de Fluidos como a Lava: De forma a animar fluxos de lava [45] baseia-se numa extensão do modelo de partículas suaves (*Smoothed Particle Model*).

Este método tem o controle sobre a evolução das dinâmicas da lava ao longo do tempo, fornecendo a cada partícula um parâmetro de viscosidade, que depende da sua temperatura atual. Durante toda a simulação, graças às transferências de temperatura na

superfície e interior do fluxo do fluido, a temperatura diminui e é nesses momentos que se verifica a transição de um comportamento de baixa viscosidade para um de viscosidade mais elevada. Ver Figura2.14.

Para além destas extensões, existem os mais variados esquemas que usam o ponto de vista de Lagrange, alguns deles são os métodos de Vórtice sem malha (*mesh free vortex methods*) descritos por Yaeger et al. [53] 1986 e Gamito et al. [15], mais recentemente por Angelidis et al. [2] e Park e Kim em [37], mas o mais comum, como visto, é o método de Hidrodinâmicas de partículas suaves (SPH) com todas as suas extensões.

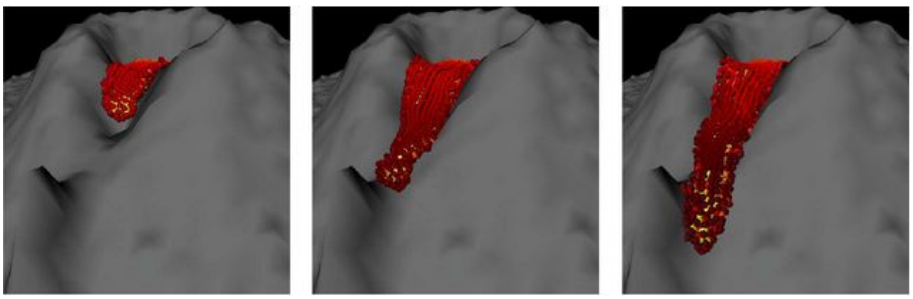


Figura 2.14: Exemplificação da variação de temperatura através da cor das partículas. Simulação composta por 3000 partículas que se espalharam ao serem expelidas pelo vulcão [45].

2.2.4 Smoothed Particles Hydrodynamics

Inicialmente o método chamado de Hidrodinâmica de partículas suaves foi desenvolvido por Gingold e Monaghem [17] e Lucy [31] em 1977 com a intenção de serem aplicados para resolver problemas da astrofísica. Apesar disso, estes métodos rapidamente começaram a ser utilizados nas mais diversas áreas de investigação, como foi o caso de Balística, Vulcanologia e Oceanografia como métodos computacionais capazes de simulação de fluidos e mecânicas.

Sendo SPH um método de Lagrange, propriedades como a Advecção de fluxos já lhe são inerentes, propriedade esta que consiste no acompanhamento da evolução de determinados elementos do fluido ao longo do tempo e espaço. Para além disso, Joseph Monaghan, explica que códigos SPH podem ser implementados de forma a que também lhe seja inerente a conservação da massa, momentum, energia e de forma semelhante (a não ser que explicitamente adicionado) eles também podem conservar a entropia. Apesar de, como dito no final da secção anterior, os SPH serem dos métodos mais comuns entre os métodos baseados em partículas, não significa necessariamente que sejam usados na sua forma original. Métodos como o SPH podem ser vistos como sendo bastante flexíveis, mas isto apenas para resolver fluxos de fluido ditos como Compressíveis, o que para simular fluidos incompressíveis, como a própria água, pode-se tornar penoso.

Para isso, foram propostas várias extensões que apareceram para facilitar a representação de tais fluidos. Uma destas formas é apresentada em [41], onde é forçada a incompressibilidade recorrendo-se a um esquema de correção de previsões (*Prediction-correction scheme*) para determinar a pressão das partículas. Para isto, é propagada pelo fluido a informação da flutuação da densidade e a pressão é atualizada até que seja alcançada a densidade desejada. Deste modo, é evitado todo o poder computacional necessário para calcular equação de Poisson da pressão, mantendo a possibilidade de, ao simular, recorrer a grandes intervalos de tempo. De forma a provar o poder desta abordagem, é realizada a comparação com uma das soluções mais comuns até à data, as *weakly compressible* SPH [5], de onde foi obtida uma melhoria bastante significativa na sua performance, como pode ser verificado pela análise da Figura 2.15.

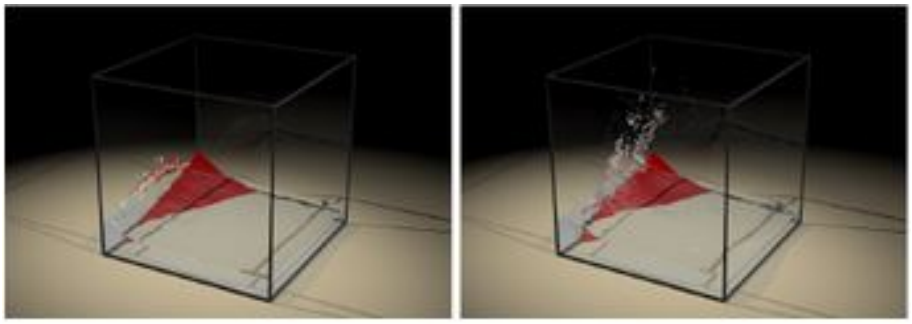


Figura 2.15: Comparação entre *Weakly compressible* SPH (esquerda, 17k partículas) e o metodo sugerido, *Predictive-Corrective Incompressible* SPH (direita, 100k partículas) e com tempos de computação iguais [41].

2.3 Simulações em Tempo Real

Como tem vindo a ser demonstrado no decorrer das secções anteriores, *Computational Fluid Dynamics* já é uma área bastante investigada, contudo ainda existem muitas dúvidas e problemas na área. Uma das principais razões para isto é a complexidade do comportamento do fluido, que pode comportar diversos fenómenos, tais como convecção, difusão, turbulência ou até mesmo tensão superficial. De maneira a computar fenómenos com um grau de complexidade acentuado como o destes, tipicamente recorre-se à divisão de duas etapas. Primeiramente é realizada a sua simulação de forma isolada, e só depois desta estar concluída é que se procede à visualização do resultado, o que impossibilita assim a interação em tempo real com outros obstáculos. Assim se deu origem ao desenvolvimento de simulações de fluidos em tempo real, que por sua vez vieram abrir portas a uma elevada variedade de aplicações.

Um dos mercados que rapidamente surge a demonstrar grande interesse nestas técnicas, é de facto, o dos videojogos, que com o continuo crescimento do poder computacional das consolas de nova geração e dos computadores, começa a entrar no universo das aplicações em tempo real, tais como jogos em três dimensões. Apesar de já haver numerosos

avanços, continua a existir um grande caminho a percorrer para que efeitos simulados em tempo real, que tipicamente devem funcionar a 40-60 *frames* por segundo num dispositivo separado, consigam alcançar a qualidade que já existe nas simulações que podem ser calculadas à priori, para os filmes, por exemplo.

Apesar do que foi mencionado no decorrer do capítulo 2.2, em que por norma métodos de Lagrange são preferíveis em comparação com os de Euler no que toca a simulação em tempo real, não significa que estes não existam. Alguns investigadores têm vindo a demonstrar métodos com a capacidade de modelar simulações de líquidos em 3D com capacidade de correr em tempo real, de forma Euleriana. Em [9] é apresentado um método para simulação de fluidos de Euler, onde é realizada a redução do tempo de computação ao recorrer a uma abordagem híbrida das representações em rede, composta por células normais (de forma cúbica) por cima de uma camada de células mais altas. Desta forma, é possível representar o fluido acima de determinado terreno sem consumir demasiada memória e poder computacional, focando-se assim em áreas que sejam de maior importância. De maneira a acelerar a simulação, é realizada uma otimização desta representação para uma implementação em GPU do *fluid solver*. Adicionalmente é proposta a modificação do *solver* e um algoritmo especificado para representação em multi-rede para resolver a equação de Poisson, e do mesmo jeito, manter a simulação estável para com intervalos de tempo maiores.

Com métodos como este, fica clara a sua capacidade de representar simulações em tempo real de líquidos tridimensionais de larga escala. Em contrapartida, se estivermos perante um jogo em que apenas uma pequena porção do mundo contém esse fluido, técnicas como esta podem tornar-se excessivas, uma vez que o poder computacional escala com o tamanho da rede e não da quantidade de fluido que a mesma contém.

Assim surgem, as técnicas de Lagrange que para simulações em tempo real, tendem a ser mais rápidas que as de Euler, pois não contendo uma rede, o fluido pode circular livremente pelo ambiente de jogo.

2.3.1 Necessidades das Aplicações

Quando se está perante conversações sobre aplicações interativas, como videojogos com pouco sucesso ou que apareçam a desiludir os consumidores, as duas principais causas tendem a ser, ou problemas técnicos, mundialmente conhecidos como *Bugs*, ou problemas de instabilidade dos *frames* por segundo. Uma das causas para tal acontecimento pode passar por uma má otimização da aplicação.

De forma a prevenir que situações como esta aconteçam [8] sugere que para conceber um algoritmo de simulação que se adeque a aplicações em jogos digitais interativos, é preciso que certas características sejam mantidas:

Computações Baratas: Como dito anteriormente em 2.3 um jogo de computador deve ser executado a pelo menos 40-60 *frames* por segundo. Entre cada *frame* devem ser calculadas várias tarefas deixando assim um curto espaço de tempo para o cálculo da simulação. Com esta informação em mente, advém a necessidade de utilizar algoritmos que contenham um poder de computação mais baixo. Bridson adiciona ainda que apesar das tarefas de simulação de físicas estarem a transitar dos CPUs para as GPUs a importância de se recorrer a algoritmos mais rápidos é mantida, já que possibilita que cada vez mais objetos possam ser simulados em tempo real.

Baixo Consumo de Memória: Esta necessidade aparece, pois, ao contrário das simulações pré-simuladas, onde era possível simplesmente comprar a memória que for necessária para simular o efeito pretendido. Quando estamos perante computadores e consolas, a memória que estas podem conter está limitada e tem de ser dividida entre as várias tarefas.

Estabilidade: Em simulações pré-computadas pode ser usado um tamanho de passos adaptativo (*adaptive time-stepping*) em situações em que ocorra um problema de estabilidade. Por outro lado, os jogos executam a um rácio de *frames* fixo. Por conta deste facto, o método de simulação tem de ser estável, para um dado tamanho de passo, independentemente do que possa acontecer.

Resultados Plausíveis: Como se pode esperar, não é propriamente possível reduzir a complexidade espacial e computacional de forma acentuada e em simultâneo melhorar a estabilidade, sem abdicar de alguma qualidade dos resultados. Desta forma, o que se requisita de uma simulação em tempo real é que seja visualmente plausível.

2.3.2 Técnicas para Simulações em Tempo Real

Analisando as restrições mencionadas em 2.3.1 e as menções do capítulo 2.3, é notável que, de facto, não existe uma abordagem que seja única e suficiente para qualquer situação que se apresente. Assim sendo, entre as várias abordagens que se pode tomar, algumas das mais comuns são:

2.3.2.1 Procedural Water

Um método que funcione de forma *procedural* invés de fazer as simulações dos efeitos físicos impostos no fluido, este limita-se a fazer apenas a animação desses efeitos. Por norma restringe-se à parte visual da superfície do fluido. Em [21] é apresentado um esquema para criar uma animação interativa e mostrar as ondas do oceano, longe da costa. Um método como este, pelo facto de não precisar de computar as simulações físicas, é capaz de produzir performances em tempo real. Á conta de não precisar de cumprir

diretamente as físicas, qualquer método pode ser admissível desde que cumpra com os efeitos visuais pretendidos, abrindo assim muitas opções para os desenvolvedores.

A grande vantagem de métodos como este, é que por não serem baseados nas físicas, são bastante fáceis de controlar, o que é útil para a criação de jogos e filmes. Por outro lado, quando se fala de interações com uma *boundary* ou objetos, isto torna-se muito difícil de conceber.

Devido a esta incompatibilidade, se a água estiver em contacto com rochas ou com a costa, esta não provocaria nenhum efeito, fazendo com que a água se limitasse a penetrar essas superfícies de forma imprópria e perturbadora ao olhar. Na prática moderna de efeitos visuais, os artistas limitam-se a cobrir estes problemas com ondas desenhadas à mão, ou recorrendo a pré-simuladores caros computacionalmente, para tratar as áreas problemáticas.

Tendo isto em consideração em [24], de forma a eliminar esta necessidade de adicionar os efeitos à mão, é apresentada uma nova estratégia, onde, de forma generalizada é feita uma transformação das trajetórias das ondas, ao qual se deu o nome de *Wave Cages*. Estas *Wave cages* são desenhadas com o objetivo de deformar as *Procedural Waves* originais, de forma que estas satisfaçam *Solid Boundary Conditions*, para evitar a penetração com outras superfícies, mantendo o movimento original das ondas em mar aberto.

As *Wave Cages* são limites desenhados através de um Elipsoide que é dimensionado, baseado na sua aproximação com um terreno, com a qual não pode ser intersetado. Ao confinar o movimento das ondas a estas elipsoides é possível garantir que as ondas não penetram o fundo do oceano, nem os obstáculos ao seu redor. Na Figura 2.16 é possível ver o funcionamento das *Wave Cages*.

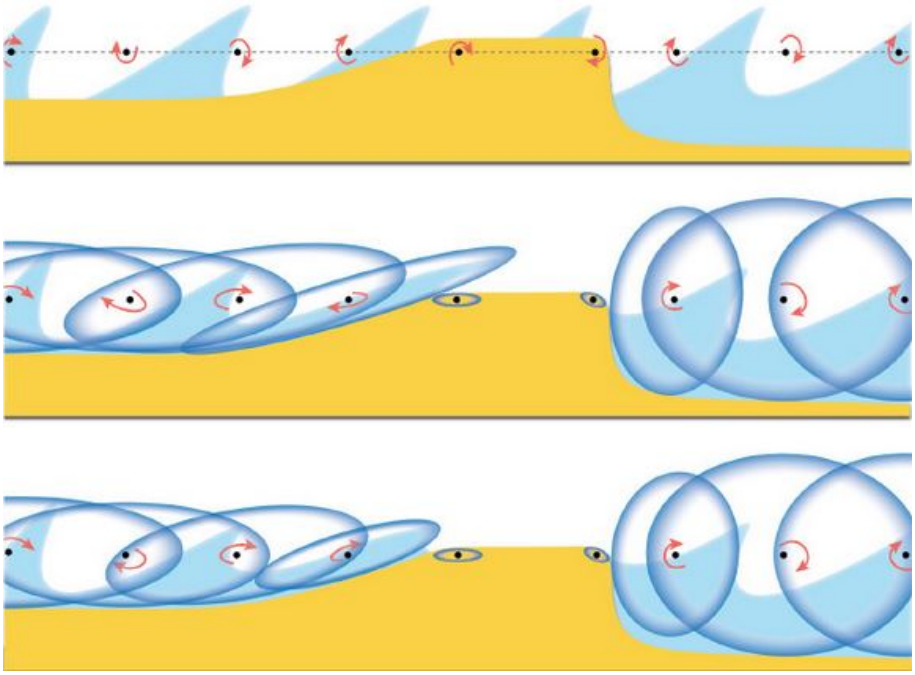


Figura 2.16: Funcionamento das *Wave Cages* [24].

2.3.2.2 Heightfield Approximations

Simulações baseadas em *Height fields*, geralmente consistem em um grupo de cubos da mesma largura, organizados de forma a originar uma área quadrada. Ao contrário dos métodos em *Procedural water*, estes geralmente implementam os aspetos físicos da simulação de fluidos. Contudo, apenas são utilizados se só for de interesse, a animação da superfície em duas dimensões do fluido, pois seria muito penoso simular um fluido por inteiro (tridimensional).

Estando perante esta situação ótima, apenas a superfície é representada como uma função em duas dimenssões/*hightfield* e animada usando uma equação de ondas em duas dimensões, aumentando assim a velocidade da simulação. Outra vantagem desta abordagem é que funciona muito bem com a implementação das texturas em *shaders*. Todavia, esta abordagem apresenta uma grande desvantagem, pois ao simular as ondas apenas controlando a velocidade vertical de cada cubo, é impossível simular a quebra de ondas sem o auxílio de outras técnicas. O costume é combinar a *Heightfields* com algum sistema de partículas, que por sua vez é ativado sempre que for necessário 2.3.2.3.

2.3.2.3 Shallow Water

Apesar dos avanços que têm vindo a demonstrar um progresso significativo no âmbito das simulações de fluidos, nomeadamente a estabelecer a precisão visual, muitos destes, não são propriamente aplicáveis em ambientes de interação em tempo real. A dificuldade que estas aplicações, como os videojogos impõem, é a larga quantidade de computações necessárias para resolver o movimento de um fluido 3D. Perante cenários como este, surgiu a ideia, que de forma a aumentar a performance de simulações de fluidos de grande área, pode reduzir-se o problema de um cenário em 3D para apenas duas dimensões com *high field*, como pode ser visto na Figura 2.17.

Uma abordagem destas mostra-se como sendo muito promissora em situações em que se esteja perante um fluido que apenas a sua superfície é de facto necessária. Diante estes cenários, surgiram então, as equações de *Shallow Water*. Este método promete ser uma alternativa a *The wave equations e High fields*, resolvendo as limitações que estes modelos apresentavam quando deparados com fenómenos como remoinhos e advecção de objetos flutuantes. Outra vantagem das equações *Shallow Water*, é que ao contrário das *The wave equations*, estas ainda são capazes de inundar áreas que estivessem secas até ao momento [28].

As *Swallow water equations* representam uma versão simplificada das Navier-Stokes 3D, sendo das mais apropriadas para a reprodução de líquidos com uma grande área, como lagos e oceanos. Estas são duas equações, uma responsável por representar a conservação da massa e outra para a conservação do momentum [28].

Contudo, apesar de representarem ondas suaves mantendo todas as propriedades da superfície livre do fluido (*free surface fluid*), quando é apresentado um fluido com ondas mais complexas e agressivas, estas deixam de ser capturadas por um modelo tão reduzido

como este. Com isto em mente, uma equipa de investigadores et al. [49] propôs uma nova técnica para aprimorar simulação de líquidos por *Height field* baseada em conjuntos de partículas de forma a criar efeitos da quebra das ondas que fossem convincentes. Contudo, neste estudo, pela quebra das ondas se tratar de um processo muito complicado e difícil de entender completamente, não foi o foco principal criar uma simulação completa destes fenómenos, mas sim, em captar as suas principais características visuais. Na Figura 2.18 pode ser observado o resultado da quebra de ondas.

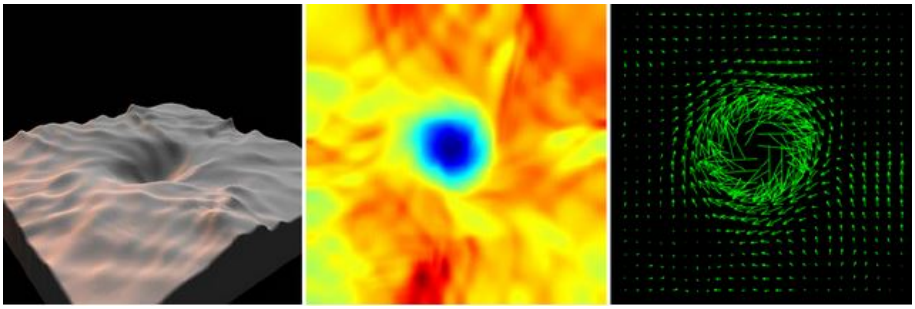


Figura 2.17: Output de um modelo de *Shallow water*. Na esquerda está representada a força aplicada pelo utilizador e à direita estão os respetivos campos de densidade e velocidade [28].

Figura 2.18: Na figura pode ver-se um personagem de um jogo a fazer surf enquanto as ondes se quebram em tempo real [49].

2.3.2.4 Smoothed Particles Hydrodynamics em Tempo Real

Retornando novamente às *Smoothed Particles Hydrodynamics*, de Muller et al. [34] mostrou que métodos SPH eram bem capazes de criar simulações de fluidos interativas com viscosidade e tensão de superfícies, em tempo real [22]. Em [34] é proposto um método interativo com foco em simulação de fluidos, como extensão de [11], onde é utilizado o SPH para animar corpos facilmente deformáveis. Para este fim, é feita uma derivação dos campos de viscosidade e pressão, diretamente das equações de Navier-Stokes, e proposto um modelo para a tensão das forças à superfície. Em prol de obter interatividade, de forma estável, precisa e rápida, é apresentado um novo design para

os *Smoothing Kernels* [8]. Como pode ser observado na Figura 2.19 os resultados obtidos não atingem o mesmo grau de realismo de animações computadas à priori, contudo, ao serem obtidos de uma simulação a correr de forma interativa, os resultados podem ser considerados promissores.

Figura 2.19: Simulação a 5 *frames* por segundo, com 3000 partículas [34].

Todavia, esta abordagem apresenta grande dificuldade na simulação de fluidos incompressíveis (ver melhor na secção 2.2.4), com a intenção de resolver este problema, técnicas como *Weakly compressible* SPH por [5] e *Predictive-corrective incompressible* SPH por [41] são propostas. Apesar do sucesso e das melhorias na sua eficiência, o tamanho dos intervalos de tempo, permaneciam como uma limitação ao desenvolvimento de aplicações em tempo real para este tipo de fluidos. Em 2013, Macklin e Müller, apresentam em [32] um *Solver* de densidade interativo integrado na *framework* de dinâmicas baseadas na posição (PBD - *Position Based Dynamics*). Realizando a formalização e a resolução de um conjunto de restrições posicionais, que impõem densidade constante, este método é capaz de resolver fluidos incompressíveis e a convergência, dos *solvers* SPH mais modernos até ao momento, mas herdando a estabilidade que os métodos de dinâmicas baseadas em posições geométricas garantem, tornando assim possível, a utilização de intervalos de tempo suficientemente grandes para aplicações em tempo real. Para além disto, ainda é incorporado um termo de pressão artificial, que por sua vez aperfeiçoa a distribuição das partículas (ver Figura 2.20), cria tensão de superfície, e reduz os requisitos impostos pelas vizinhanças, que exerciam várias limitações aos SPH tradicionais.

2.4 Interação e Simulação Gráfica

Nesta secção é feita uma pequena introdução do que são motores de jogos e de como estes funcionam. Para além disso, também serão apresentados alguns videojogos e como estes implementaram simulações de fluidos na sua produção.

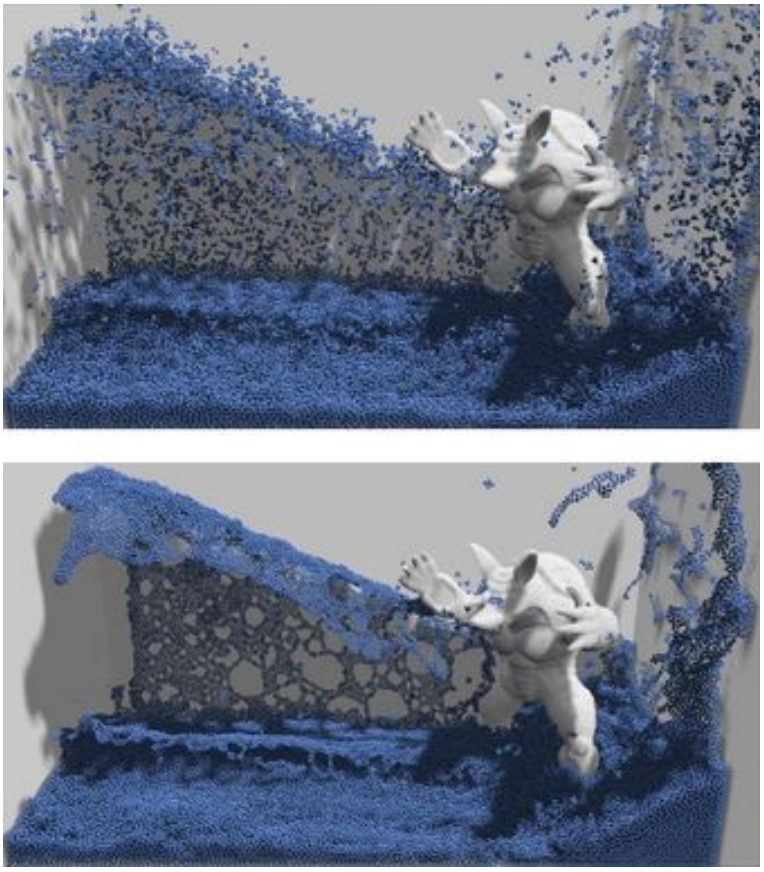


Figura 2.20: Cima: Aglomeração de partículas devido às deficiências da vizinhança, Baixo: Com o termo de Pressão artificial é notável a melhoria na distribuição das partículas e da tensão da superfície [32].

2.4.1 Motores de Jogo

Um motor de jogos é uma *software framework* desenvolvida principalmente para a criação de videojogos. A sua utilização é bastante benéfica, uma vez que, geralmente pode incluir *libraries*, como o caso do ML-Agents Toolkit (que consiste num projeto *open-source* que permite que jogos e simulações contenham ambientes para o treino de agentes inteligentes) e o mais variado tipo de plugin para facilitar o desenvolvimento [50].

Uma das principais funcionalidades fornecidas por um motor de jogos, é cuidar da renderização gráfica (*renderer*) para imagens 2D e 3D. Para além disto, ainda é capaz de computar outras tarefas, tais como, inteligência artificial, deteção de colisões entre objetos de jogo e gestão de memória.

Outro atributo de grande importância nos motores de jogos é a abstração em relação à plataforma, de maneira que é possível correr o jogo, para várias plataformas diferentes, sem ter de efetuar grandes alterações ao código fonte do jogo.

2.4.1.1 Contextualização Histórica

O termo de "motores de jogos"surgiu por volta de 1990, com o aumento de popularidade dos jogos em 3D, especialmente por conta de jogos como, Quake e Doom. A demanda por jogos como estes continuava a crescer, mas consequentemente, os custos no desenvolvimento de simulações mais reais tornavam-se de tal forma insuportáveis que se tornava impossível, para os criadores, recuperar todo o seu investimento, surgindo assim, a emergência pela criação de motores de jogo [29]. Com isto, o crescimento de *softwares* como este, continua até aos dias de hoje, havendo uma variedade enorme de motores de jogos. Estes avanços continuam a ser de grande importância, já que abriu a possibilidade de qualquer individuo criar o seu próprio videojogo, sem precisar de grande investimento.

2.4.2 Simulação de Fluidos em Videojogos

É fácil de argumentar que grande parte da diversão e do prazer em jogar videojogos tem origem na qualidade e capacidade de interação com o mundo do jogo. E para isso ser possível, é necessário um sistema de físicas de corpos rígidos (*rigid bodies*) [25]. Tendo isto em mente, fica crucial ter um ambiente de jogo que seja imersivo e que tenha componentes para que o jogador aprecie a sua experiência enquanto o joga. Obviamente que para alcançar isto, pode haver várias abordagens diferentes, mas de facto, algo que se tem vindo a tornar cada vez mais requisitado nestes jogos, são as simulações de fluidos. Para além de muitos efeitos visuais, estas possibilitam ainda que seja possível a implementação de fenómenos como o vento, ondas no oceano, ondas criadas por barcos a navegar ou até uma simples simulação de água a encher um copo.

Tendo tudo isto em consideração, fica fácil de perceber porque razão estas simulações têm ganho tanto protagonismo no cenário do desenvolvimento de videojogos, e como tal, têm sido criadas as mais variadas técnicas de simulação de fluidos para os vários fenómenos que se queira apresentar num jogo. Assim sendo, nesta secção serão apresentados alguns dos maiores jogos da atualidade e as técnicas de simulação de fluidos que foram usadas no seu desenvolvimento.

Uncharted 3: Drake's Deception : Jogo lançado em 2011, desenvolvido pela Naughty Dog, e trata-se de um jogo de ação e aventura em terceira pessoa. Neste jogo foram feitos vários tipos de água, tais como: cascatas, poças, o oceano e aquela que vai ser apresentada aqui, inundações. Em [10] por Eben Cook, é explicado que inicialmente foi feita uma simulação de fluidos através de um percurso, dentro de um *Software* chamado Houdini, que possibilitou uma melhor visualização dos *timmings*, fluxos e a quantidade de água necessária, em relativamente pouco tempo (ver Figura 2.21). Todavia, esta simulação não foi usada apenas para pré-visualização, pois acabou por ser usada para a criação da *In-game mesh*, como ponto de partida de como as partículas deviam ser espalhadas, e ainda usaram a *data* dessa simulação para mover os obstáculos (os *rigid-bodies*) de forma a que tudo combinasse corretamente. O primeiro problema a surgir, foi na criação da

mesh já que por se tratar de um fluido em movimento, a superfície teria de estar em constante transformação. Para isto, foram apresentadas várias alternativas, mas nenhuma se adequava à situação. Então, a opção tomada foi recorrer a um método de força bruta onde se recorreu à utilização de *Skeletal meshes* (que era bastante otimizado para o motor utilizado) que por sua vez, foi otimizado ao aproveitar o ângulo da câmara, para invés de utilizar uma *mesh* de quadrados, usar uma versão mais alongada, com retângulos, e assim, reduzir em dois terços a quantidade de pontos, e por sua vez de *joints* necessários para a mesma. Para extrair a superfície da simulação de fluidos, foi usada a técnica de *ray cast* onde ao invés de fazer o *cast* ao longo das normais (do plano), os "raios"foram feitos em direção à câmara, e assim evitar paredes lisas na representação do fluido (Analisar Figuras 2.22 e 2.23).

Por fim, para criar um efeito mais agressivo, foram criadas partículas, para o qual foram usados 8 tipos de partículas diferentes e 31 emissores de partículas, animados *frame* a *frame*.

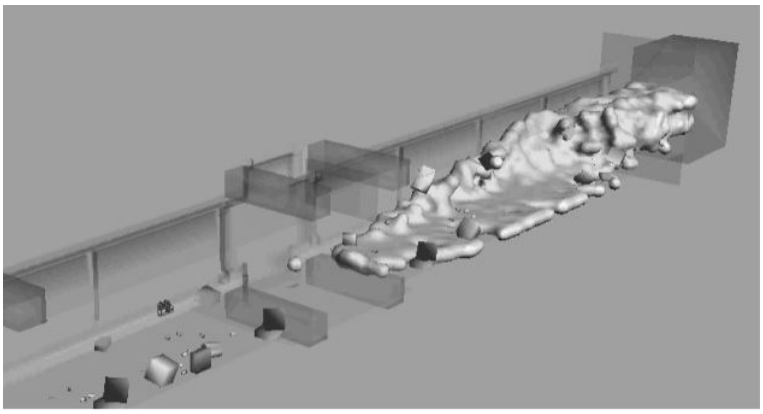


Figura 2.21: Simulação de fluidos criada no Houdini [8].

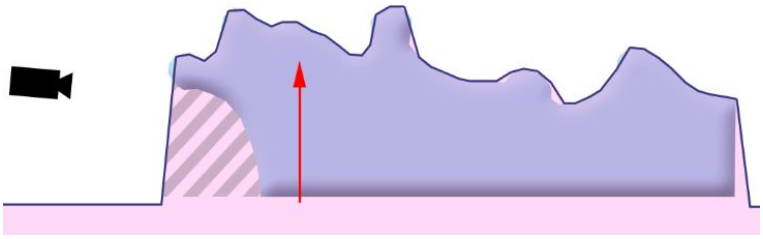


Figura 2.22: Criação da *mesh* usando *casting* ao longo da normal da superfície [9].

[8]Nome: GDCVault, Link: https://www.gdcvault.com/play/1015453/Creating-the-Flood-Effects-in, último acesso: fev 2022

[9]Nome: GDCVault, Link: https://www.gdcvault.com/play/1015453/Creating-the-Flood-Effects-in, último acesso: fev 2022

[10]Nome: GDCVault, Link: https://www.gdcvault.com/play/1015453/Creating-the-Flood-Effects-in, último acesso: fev 2022

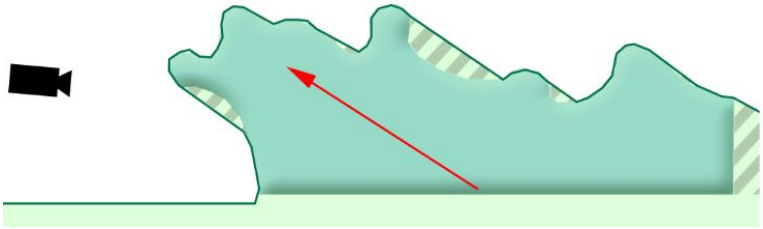


Figura 2.23: Criação da *mesh* usando *casting* em direção à câmara [10].

Sea of Thieves : Este é um jogo de ação e aventura, lançado em 2018 pela Rare. Foi desenvolvido no Unreal Engine 4 com o objetivo de conter vários elementos dinâmicos e que fosse capaz de funcionar desde GPU's integrados em computadores portáteis, a computadores de última geração. Em [1] é explicado que para implementar a simulação da água sob superficial, foi usada a técnica *Fast Fourier Transformation (FFT)* de Tessendorf [48] onde é feita uma explicação de como criar uma superfície mapeada pelo deslocamento com o método de *Hight-field* para criar ondas no mar, de grande detalhe. Para além disso, ainda são apresentados vários efeitos para a ambientação debaixo de água, tal como fazer a sua modelação e *renderização*.

Com o objetivo de atingir o aspeto e efeitos desejados, foram realizadas algumas estratégias personalizadas pela equipa de desenvolvimento. Para a cor da água foram usadas aproximações de dispersão *scattering approximations* e fez-se uma mistura entre as cores da água profunda e da água sob superfície, baseadas na combinação do ângulo de visão, direção do sol e da máscara dos picos de ondas. Por sua vez, essa máscara, foi gerada através dos *FFT choppiness vertex offsets* que dependendo do tamanho do corte, resultará os picos com mais ou menos impacto da sob superfície. A espuma foi gerada nos picos de onda, onde também foram utilizadas, como inspiração, as técnicas apresentadas em [48], acabando por ser utilizado em torno de objetos que de alguma forma intersetem a superfície da água. Para deixar o jogo mais imersivo, dependendo do tempo, no caso de tempestade, por exemplo, os artistas criaram modificações na textura para que seja possível adicionar mais ou menos espuma.

Por fim, para criar efeitos como, salpicos da água a bater no convés do navio, foi realizada uma simulação de superfície da água em GPU baseada em [33]. Este método, consiste em modelar fenómenos como a erosão hidráulica, executado a frequências interativas, e é baseado no campo de velocidade (*velocity field*) da água corrente, que é criado com um modelo de *Shallow Water fluid*. Dessa forma, o campo de velocidade é usado para o cálculo do processo de erosão, deposição e transporte dos sedimentos. Ademais, com o objetivo de representar a interação dos personagens a passar através de uma cascata ou da espuma em redor dos seus pés (ao mover-se numa poça), projetou-se, o *buffer* de profundidade (*deph buffer*) da perspetiva da câmara para *mesh* da superfície, no espaço de textura da sua simulação de *Shallow Water*.

3

Arquitetura de Sistema de Simulação de Fluidos

A solução proposta é desenvolver um sistema de simulação de fluidos, tridimensional e com interação, desenvolvido num motor de jogos (Unity), com o objetivo de ser rápido e eficiente, e que também funcione em projetos de interação em que se precise da simulação de fluidos, e que seja preciso e convincente, num ponto de vista mais científico. Para isso, serão criados dois tipos de simulação distintos. Um baseado em partículas de Lagrange, que por sua vez, deverá ter dois métodos de otimização distintos, para uma forma mais eficiente de atingir a interação com o fluido em tempo real. E um segundo, também baseado em partículas, mas de uma forma híbrida, ao trabalhar em conjunto com uma *Spatial hash table* perspetivando atingir resultados mais rápidos e precisos cientificamente. Para além disso, deve ser possível para o utilizador, alterar os parâmetros necessários, para atingir a simulação pretendida.

Para se entender melhor de que forma esta solução será implementada, neste capítulo, será apresentado na secção 3.1 uma pequena descrição do que se pretendeu atingir, seguida pela caracterização dos principais requisitos do nosso sistema 3.2 e examinar a estrutura do mesmo, juntamente com a arquitetura do algoritmo de simulação utilizado 3.3. De seguida, são apresentadas as principais ferramentas e técnicas utilizadas 3.4 e por fim, na secção 3.5 é possivel analisar mais detalhadamente os conceitos e formulações utilizados no desenvolvimento do nosso sistema de simulação de fluidos.

3.1 Descrição Geral

O cerne do desenvolvimento desta dissertação esteve em criar um simulador de fluidos, em tempo real e com interação, num motor de jogos (Unity), regido pelas normas estabelecidas no estudo das CFD (Computational Fluid Dynamics), focando-se assim, na fisíca dos fluidos. Para isso, o método implementado é fundado pelas conhecidas, SPH *Smoothed-particle hydrodynamics*, composto por quatro abordagens quanto à computação de interação de particulas. O sistema implementado, trata-se de um ambiente 3D de

simulação de fluidos (líquidos), com a capacidade de representar múltiplos fluidos independentes um do outro e em simultâneo, em diferentes tipos de ambiente, com controlo sob as várias propriedades de um fluido. Assim, é possivel atingir diferentes tipos de interação como salpicos, ondas, ou interação com outros objetos (podendo esta interação, conter um bom espaço para melhorias).

Para assegurar que o sistema está desenvolvido corretamente, foram necessários diferentes cenários que testassem as várias propriedades e comportamentos que um fluido pode apresentar. Assim, com recurso a estes cenários, foram realizados diversos testes a cada um dos algoritmos implementados, que por sua vez, foram comparados entre si.

3.2 Requisitos do Sistema

Nesta secção, será apresentada uma lista de requisitos, que serão necessários no desenvolvimento e na versão final do sistema.

Os requisitos do sistema são:

1. ***Renderizador em Tempo-Real:*** O motor de jogos precisa de conter um *renderer* em tempo real para que seja possivel renderizar a simulação enquanto esta corre.

2. ***Controlador de Cenas:*** O motor de jogos precisa de ter um *scene manager* para que seja possivel criar e editar ambientes de jogo.

3. ***Materiais:*** O motor de jogos precisa de ter acesso a diferentes tipos de materiais de modo a que seja possível atribuir diferentes texturas aos diferentes tipos ou estado das partículas.

4. ***Sistema de estatísticas:*** O motor de jogos precisa de conter um sistema de estatísticas que permita ao utilizador, analisar as diferentes estatísticas produzidas pela simulação, em tempo de processamento(*Run-time*).

Quanto aos utilizadores, os requisitos são:

1. ***Controlo de parâmetros:*** O utilizador deve ter acesso a um sistema de controlo de parâmetros para que seja possivel configurar as definições da simulação, como viscosidade ou tipo de fluido.

2. ***Controlo do tipo de algoritmo:*** O utilizador deve ser capaz de selecionar a otimização/algoritmo de simulação que pretende utilizar.

3. ***Controlo sobre os emissores:*** O utilizador deve ter a possibilidade de adicionar e remover emissores de fluidos quando desejar.

4. ***Adicionar e remover obstáculos:*** O utilizador deve poder adicionar e remover obstáculos ao ambiente de jogo onde está a ocorrer a simulação.

5. ***Controlo sobre o vídeo:*** O utilizador tem de ter a possibilidade de alterar entre as várias câmaras, de forma a ter um melhor ângulo de visão para o que pretende observar.

6. ***Ver estatísticas:*** O utilizador deve ser capaz de ver as estatísticas da simulação, para saber quantas particulas estão em Cena, conhecer os parâmetros atuais da simulação e a taxa de *frames* por segundo.

7. ***Controlo sobre a simulação:*** O utilizador deve ser capaz de controlar o decorrer da simulação, podendo pausar, reiniciar e terminar a simulação.

8. ***Controlo sobre o modo de visualização:*** O utilizador deve ser capaz de trocar entre diferentes modos de visualização.

3.3 Arquitetura do Sistema

Nesta secção serão apresentados diversos diagramas com as diferentes componentes responsáveis por descrever o funcionamento do sistema de simulação de fluidos implementado. Para além disso, ainda será apresentado o *pipeline* em ciclo com as correspondentes componentes e conexões, que mapeiam os cálculos das físicas e transferência de informação, sendo assim, responsável pela computação das fisicas do fluido a simular.

O sistema implementado é constituido principalmente por quatro etapas. Inicialmente é necessário criar um ambiente onde executar a simulação pretendida. O processo de criação destes ambientes é demonstrado no capítulo mais à frente. De seguida, é essencial que o utilizador introduza as propriedades do fluido que pretende simular, através das interfaces apresentadas como *MasterController* e *Data*, que serão detalhados mais adiante. Finalizadas estas duas etapas, o sistema fica encarregue de executar todas as computações necessárias para descrever o comportamento do fluido pretendido e gerar a simulação. Por fim, o Unity expõe todo este comportamento de forma visual, e o utilizador é capaz de observar toda a simulação gerada, analisar algumas estatísticas e interagir com o próprio fluido. Todas estas etapas podem ser resumidas pelo diagrama da Figura 3.1.

Afim de entender com mais detalhe todo o funcionamento do sistema implementado, o mesmo foi dividido em dois pontos do vista diferentes.

Arquitetura do ponto de vista do utilizador: Começando por analisar o sistema, pela perspectiva do programador que usa o sistema, o foco estará no diagrama apresentado na Figura 3.2.

Com o propósito de oferecer mais detalhe em relação ao funcionamento do sistema, decompôs-se a etapa de criação de um ambiente de simulação, resultando assim, em diferentes fases. Na fase (1), o utilizador terá de começar pela criação de um cenário de

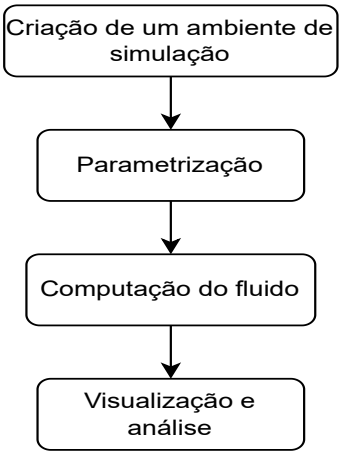


Figura 3.1: Arquitetura geral do sistema.

simulação onde reproduzir a simulação, seguido da criação dos controladores responsáveis por controlar todo comportamento e criação do fluido a simular. Para a etapa da Parametrização as fases (2),(3) e (4) indicam as diversas propriedades que devem ser preenchidas no controlador do fluido, em que numa primeira fase é necessário indicar as diferentes propriedades da simulação desejada 4.3, na segunda, são necessárias todas as propriedades do fluido pretendido 4.4, e numa quarta fase, caso se pretenda utilizar uma versão do algoritmo que precise de recorrer a uma grelha, é ainda necessário que se indique as propriedades da grelha onde será dada a simulação (Figuras 4.5 e 4.6). Finalmente, o fluido será computado pelo sistema, na fase (5), e por fim na fase (6) será apresentada a visualização da simulação, tal como, os resultados estatísticos obtidos.

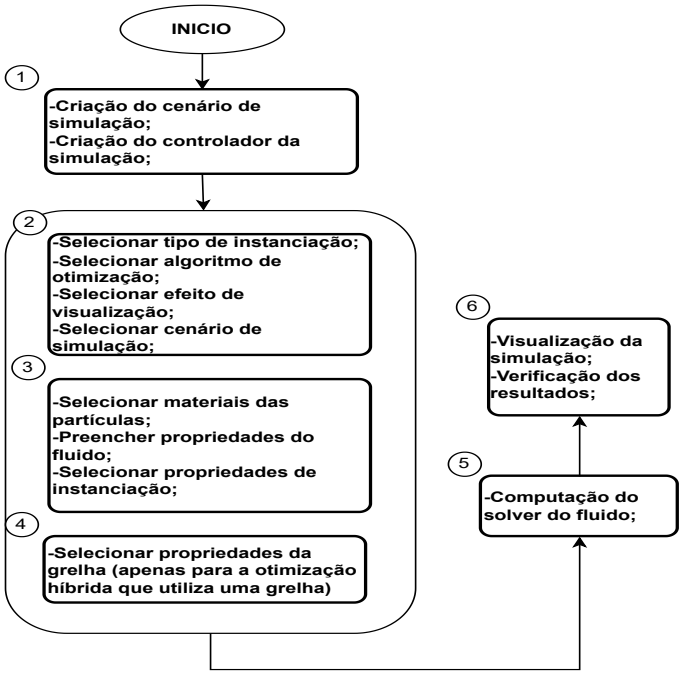


Figura 3.2: Arquitetura geral do ponto de vista do utilizador.

Arquitetura do ponto de vista do sistema: De modo a entender melhor todo o funcionamento do sistema, falta apenas entender as computações necessárias para reproduzir todo o comportamento de um fluido e de que maneira estas são executadas.

No diagrama da Figura 3.3, é apresentado o ***pipeline*** em ciclo com as correspondentes componentes e conexões, que mapeiam os cálculos de físicas e transferência de informação entre computações, sendo assim responsável pela representação da simulação do fluido. O processo de criação da simulação de um fluido, começa pela **componente (1)**, responsável por receber o input inicial, incluindo todos os parâmetros do fluido, com todas as propriedades preenchidas. Com toda a informação necessária, o sistema, procede a inicialização das partículas, executando assim, a **componente (2)**. Desta forma, são criadas as posições iniciais de cada partícula, tal como é demonstrada na **componente (3)** do diagrama. Estes vetores descrevem o estado atual do fluido a ser simulado.

Nesta etapa, o sistema já terá à sua disposição as informações necessárias para gerar o comportamento do fluido e assim, retornar a próxima iteração do mesmo. Para isso, segue-se a **componente (4)**, que consiste em calcular os valores de densidade e pressão de cada partícula, no momento da iteração de sistema atual. Por sua vez, na **componente (5)** os valores de densidade e pressão obtidos serão atribuidos às partículas correspondentes. Desta forma, nas **componentes (6) e (7)** são calculadas e atribuídas, respectivamente, as forças de pressão, viscosidade e gravidade, aplicadas em cada partícula. Com estas forças, estamos cada vez mais próximos de obter o movimento do fluido, mas não o suficiente. Para isso, é necessário obter a nova velocidade e posição das partículas, no intervalo

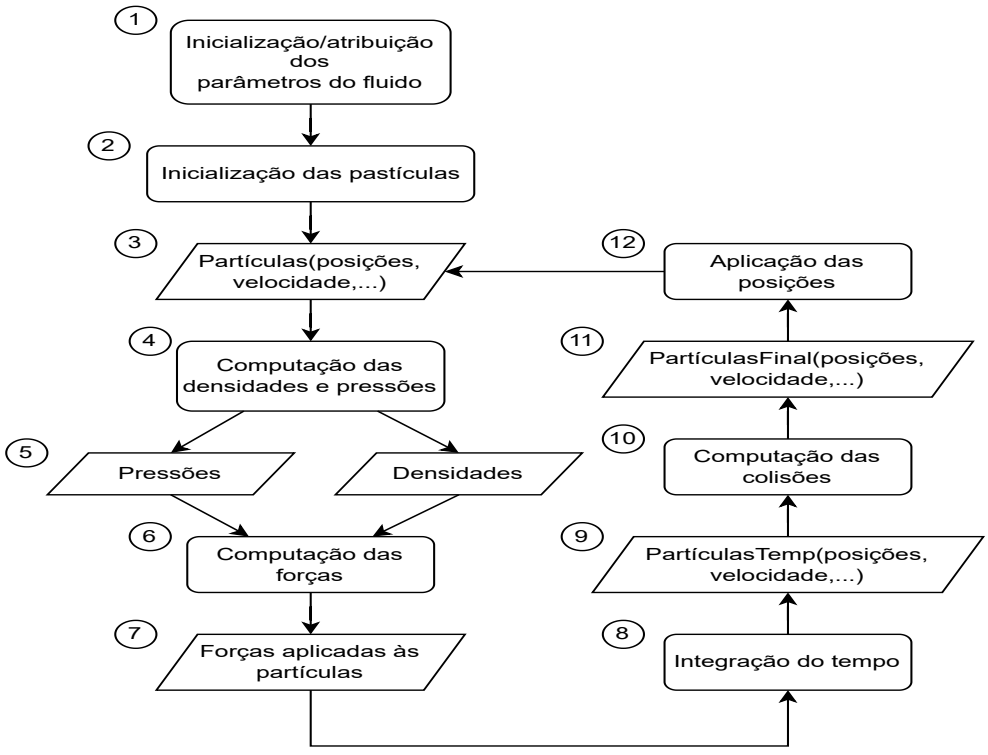


Figura 3.3: Arquitetura geral do ponto de vista do sistema.

de tempo pretendido. Desta forma, foi fundamental criar a **componente (8)**, ficando responsável por traduzir as forças obtidas na **componente (7)**, em posições e velocidades resultantes. Estes resultados originam novas versões das partículas do fluido, guardadas na **componente (9)** do diagrama 3.3.

Tal como foi demonstrado até ao momento, já foram calculadas todas as interações entre partículas que foram geradas na computação do fluido. Todavia, estes cálculos não são os suficientes para o correto comportamento do fluido, já que para além da interação entre as ditas partículas, também é necessário calcular as interações com o cenário em que o fluido se apresenta. De modo a obter estes resultados, é implementada a **componente (10)**, onde todas as colisões das partículas com o ambiente envolvente serão, por sua vez, calculadas, resultando na versão final das partículas, com todas as suas propriedades atualizadas **(componente (11))**. Para finalizar, existe uma última **componente (12)**, responsável por atribuir às partículas reais, as suas posições finais. De notar que, como o fluido se encontra em continuo movimento e interação, as posições finais calculadas em **(12)**, serão as novas posições iniciais da próxima iteração do sistema, formando assim, um ciclo de computações.

3.4 Ferramentas e Técnicas

As técnicas utilizadas para o desenvolvimento de um ambiente de simulação foram investigadas e decididas na fase inicial do desenvolvimento, tendo em consideração os requisitos do sistema na secção 3.2 e o potencial para trabalhos futuros sob este mesmo projeto. Com isto, chegou-se à conclusão que a melhor abordagem seria recorrer a simulações de fluidos baseadas em SPH descritas na secção 2.2.4. Para além disso, o SPH é uma técnica baseada em partículas relativamente nova, tendo recebido grandes melhoramentos nos últimos anos, o que em combinação com o facto de ser livre de *mesh* tem vindo a ser utilizada em vários casos de simulações industriais, para os quais, os métodos baseados em *mesh*, não fossem suficientemente eficientes. Assim, métodos baseados em SPH continuam a mostrar-se, métodos de elevada relevância até aos dias de hoje [1].

Relativamente às ferramentas utilizadas, este projeto foi inteiramente desenvolvido no motor de jogos Unity. Algo importante de referir, é que desde o princípio da investigação, que se pretendeu criar um ambiente de simulação de forma nativa, no que diz respeito às físicas do fluido, ou seja, sem o auxílio de ferramentas como: *Rigid Bodies*, *Colliders* e/ou, até mesmo, a componente de *Physics* do próprio Unity. Esta escolha foi feita para que a simulação seja meramente guiada pelas *Computational Fluid Dynamics* (CFD), com o auxílio das componentes de representação gráfica e computacional do Unity, sendo que o Unity já foi criado com a capacidade de representar um ambiente 3D em vista.

[1] Dive Solutions, https://www.dive-solutions.de/articles/sph-basics, último acesso: março 2023

3.5 Simulação com Smoothed Particles Hydrodynamics

3.5.1 Navier-Stokes

De forma geral, o SPH é uma discretização de um fluido em várias partículas, que por sua vez, terão as suas propriedades amenizadas (*Smoothed*) por um *Kernel*. Para isto, ao invés de se considerar que todas as partículas interagem com todas as restantes, o que acontece, é uma verificação da distância entre as duas partículas e se esta respeitar o *Smoothing radius* definido, propriedades como a pressão e a densidade são calculadas e aplicadas a estas mesmas partículas, caso contrário, as sua interação torna-se menosprezada.

Analisando e usando como ponto de partida o que foi anteriormente referido no decorrer da secção 2.1 as equações de Navier-Stokes exercem um grande impacto em grande parte das simulações de fluidos, e a técnica de simulação SPH é mais uma destas simulações.

No decorrer desta pesquísa, o foco foi mantido especificamente nas equações incompressíveis de Navier-Stokes, que é a versão das Navier-Stokes responsável por controlar fluidos como a água. Existem outras formas de representar estas equações, como as compressíveis equações de Navier-Stokes, que geralmente, são utilizadas para representar fenómenos Super e Hipersónicos, onde é possível obter descontinuidades nas soluções, coisa que não é possível nas equações incompressíveis. A razão para essa escolha, é que os fluidos que se pretende simular tendem a corresponder com a normal intuição de comportamento de um fluido, podendo então, ser representados por estas equações. Reformulando as equações demonstradas na secção 2.1.2 é possível chegar à equação da conservação do momentum para técnicas baseadas em grelhas (Euler) pelas formulações:

$$\rho(\frac{\delta\vec{u}}{\delta t} + \vec{u}\cdot\nabla\cdot\vec{u}) = -\nabla p + \rho\vec{g} + v\nabla\cdot\nabla\vec{u} \tag{3.1}$$

e para a equação responsável por garantir a conservação da massa, temos:

$$\frac{\delta\rho}{\delta t} + \nabla\cdot(\rho v) = 0 \tag{3.2}$$

Onde $\vec{u}$ corresponde à velocidade, ρ o campo de densidade, p o campo de pressão, $\vec{g}$ representa as forças externas e o v a viscosidade do fluido. Presentes na literatura encontram-se diversas formas de representar as equações de Navier-Stokes, mas a equação 3.1 representa uma versão simplificada para fluidos incompressiveis.

Apesar de já apresentarem um formato relativamente mais simplificado, estas podem resultar em equações significativamente mais simples, se invés de se recorrer a grelhas estacionárias, optar-se por métodos baseados em partículas (Lagrange). Isto, porque para além do número de partículas ser constante e cada partícula conter massa constante, garante-se assim a conservação da massa, o que permite a omissão da equação 3.2. As partículas também se movem com o fluido, o que significa que a derivada substancial

(Material) do campo de velocidade fica simplesmente a derivada da velocidade, em relação ao tempo, podendo assim remover-se o termo de convecção $\nabla \cdot (\rho v)$, resultando na formalização:

$$f = -\nabla p + \rho \vec{g} + v\nabla \cdot \nabla \vec{u} \tag{3.3}$$

Onde o f representa a mudança do momentum das partículas ($\rho \frac{\delta \vec{u}}{\delta t}$) determinado pela soma dos termos da pressão $-\nabla p$, viscosidade $v\nabla \cdot \nabla \vec{u}$ e atuação das forças externas $\rho \vec{g}$.

3.5.2 Pressão e Viscosidade

Até aqui foi possivel perceber, de uma forma mais abstrata, o raciocínio que existe por trás das tão conhecidas, equações de Navier-Stokes. De onde, para fazer uma aproximação com o *SPH*, é preciso aplicar as componentes de pressão e viscosidade.

$$f_i^{pressão} = -\nabla p = -\sum_j m_j \frac{Pj}{\rho j} \nabla W(r_i - r_j, h) \tag{3.4}$$

$$f_i^{viscosidade} = v\nabla^2 \vec{u} = v \sum_j m_j \frac{u_j}{\rho j} \nabla^2 W(r_i - r_j, h) \tag{3.5}$$

Este cálculo da pressão para cada particula em cada interação com outras partículas, é feito com recurso à massa (m_j), e aos cálculos intermédios da pressão (p) e do derivativo do *Smoothing Kernel* ($W(r,h)$ de comprimento h num vetor r).

A pressão p é calculada de forma simples através da constante de Gás ideal, pela equação:

$$p = k\rho, \tag{3.6}$$

onde k representa a constante de gás, e ρ a densidade , tal como sugerido por Muller et al. [34] foi utilizada a versão da equação anterior, sugerida por Desbrun et al. [11]. Desta modificação, resulta a equação dada por:

$$p = k(\rho - \rho_0), \tag{3.7}$$

onde ρ_0 representa a densidade de repouso (*rest density*) que é a densidade do fluido quando este se encontra em equilibrio. Isto significa que, se numa certa área, a densidade for superior à densidade de repouso, teremos pressão positiva, enquanto que se a densidade for inferior à densidade de repouso, teremos pressão negativa, resultando numa sucção do fluido. Com este pequeno deslocamento (*offset*) é também garantida uma simulação mais estável do nosso fluido [34].

Da mesma forma, que para a pressão, o cálculo da viscosidade em cada interação com outras partículas, é feito com recurso a massa (m_j) a velocidade das particulas u e ao laplaciano (operador de Laplace) do *Smoothing Kernel* ($W(r,h)$) de comprimento (h) num ponto (r)).

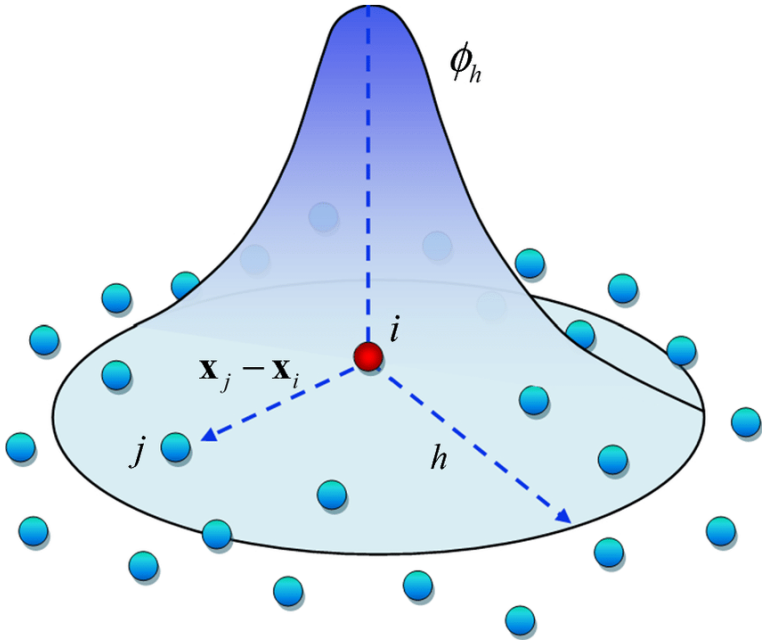


Figura 3.4: Esquema que ilustra o comportamento dos *Smoothing Kernels* para métodos de SPH no caso de um ambiente 3D [3].

3.5.3 Kernel Functions

Como mencionado anteriormente, o *Smoothed particle hydrodynamics (SPH)* é um popular método de partículas de Lagrange livre de *mesh*, que consiste principalmente na utilização de *kernel functions* ($W(r,h)$) para realizar as aproximações numéricas. Apesar de existir uma boa variedade de *Smoothing kernels* que se pode utilizar, sendo possível encontrar investigações onde se prove porquê que alguns demonstram ser mais apropriados que outros, dependendo também do tipo de simulação ou fluido que se quer obter.

Matthias Müller, no desenvolvimento da sua investigação sobre *Particle-Based Fluid Simulation for Interactive Applications* et al. [34] afirma "Stability, accuracy and speed of the SPH method highly depend on the choice of the smoothing kernels.", afirmando mais uma vez, a importância e impacto causado pela escolha de *Smoothing kernels* adequados à simulação pretendida. Estes, tendo de obedecer a algumas propriedades, como: W, ou seja, o valor obtido do *Smoothing kernel*, ter de convergir para zero, chegando mesmo a zero, ao atingir uma distância h entre as partículas, concluindo que essas partículas estão de tal forma distantes uma da outra que a sua influência uma na outra, fica nula. E em segundo lugar, W tem de chegar a 1 sobre a área de uma esfera de raio h.

Ao concluir que a escolha do *Smoothing Kernel* apresenta um papel crucial na representação precisa das propriedades do fluido e das interações entre partículas, esta escolha dependeu das propriedades e características que se pretende para o fluido a simular. Destas propriedades, o foco incidiu em garantir a qualidade visual e precisão da simulação, algo atingido pelo efeito de suavização melhorado por Müller et al. [34]. Em adição, com recurso aos Kernels de Müller, também existe a redução da criação de aglomerados de

partículas, promovendo simulações mais estáveis e precisas. Assim, com base nestes fatores, o desemvolvimento deste projeto acabou por ser bastante inspirado nos trabalhos de Matthias Müller et al. [34] especialmente quanto à escolha dos *Smoothing Kernels*.

Müller's Poly6 Kernel:

$$W_{poly6}(r,h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r^2)^3, & \text{if } 0 \leq r \leq h \\ 0, & \text{otherwise} \end{cases}, \tag{3.8}$$

onde h corresponde ao *smoothing radius* e r a distância entre as duas particulas a ser iteradas.

Este primeiro kernel foi proposto principalmente com vista no cálculo da aproximação da densidade ρ, obtendo uma curva de resultados que se demonstrou bastante satisfatória. Para além disso, algo importante de notar neste kernel é que r só aparece elevado ao quadrado, tornando o seu cálculo bastante mais eficiente por deixar de ser necessário computar a raiz quadrada ao computar a distância entre as partículas (operação esta que pode apresentar um grande custo computacional). Contudo, ao tentar computar as forças aplicadas pela pressão, com recurso a este kernel, as partículas demonstraram uma forte tendência a formar pequenos aglomerados nas zonas de alta pressão. Resultado da aproximação entre as partículas, que ocorre devido ao desaparecimento das forças de repulsão, que se anulam, pois o gradiente deste mesmo kernel tem uma forte tendência de se aproximar de zero à medida que as partículas se aproximam. Tal evento pode ser observado na Figura 3.5.

Spiky Kernel:

$$W_{spiky}(r,h) = \frac{15}{\pi h^6} \begin{cases} (h - r)^3, & \text{if } 0 \leq r \leq h \\ 0, & \text{otherwise} \end{cases}, \tag{3.9}$$

que por sua vez, ao ser calculado o seu gradiente, para assim se obter a força aplicada pela pressão, obtemos o kernel:

$$\nabla W_{spiky}(r,h) = -\frac{45}{\pi h^6} \begin{cases} (h - r)^2, & \text{if } 0 \leq r \leq h \\ 0, & \text{otherwise} \end{cases}, \tag{3.10}$$

onde h corresponde ao *smoothing radius* e r a distância entre as duas particulas a ser iteradas.

Este kernel foi obtido por Desbrun et al. [11] a fim de resolver o antigo problema do *poly6 kernel*, em que o gradiente tende para zero com a aproximação das partículas. Mantendo a condição de sempre que as partículas se afastam até um raio h tanto o kernel como o gradiente e o Laplaciano.

Viscosity Kernel:

$$W_{viscosity}(r,h) = \frac{15}{2\pi h^3}\begin{cases} -\frac{r^3}{2h^3} + \frac{r^2}{h^2} + \frac{h}{2r} - 1, & \text{if } 0 \le r \le h \\ 0, & \text{otherwise} \end{cases}, \tag{3.11}$$

que por sua vez, ao ser calculado o seu Laplaciano, para assim se obter a força aplicada pela pressão, obtemos o kernel:

$$\nabla^2 W_{viscosity}(r,h) = \frac{45}{\pi h^6}\begin{cases} (h-r), & \text{if } 0 \le r \le h \\ 0, & \text{otherwise} \end{cases}, \tag{3.12}$$

onde h corresponde ao *smoothing radius* e r a distância entre as duas partículas a ser iteradas. E tal como o *spiky kernel*, também este mantém a mesma condição.

Para além disso, de acordo com Müller ao utilizar este kernel para as computações da viscosidade, estas apresentam um aumento na estabilidade da simulação.

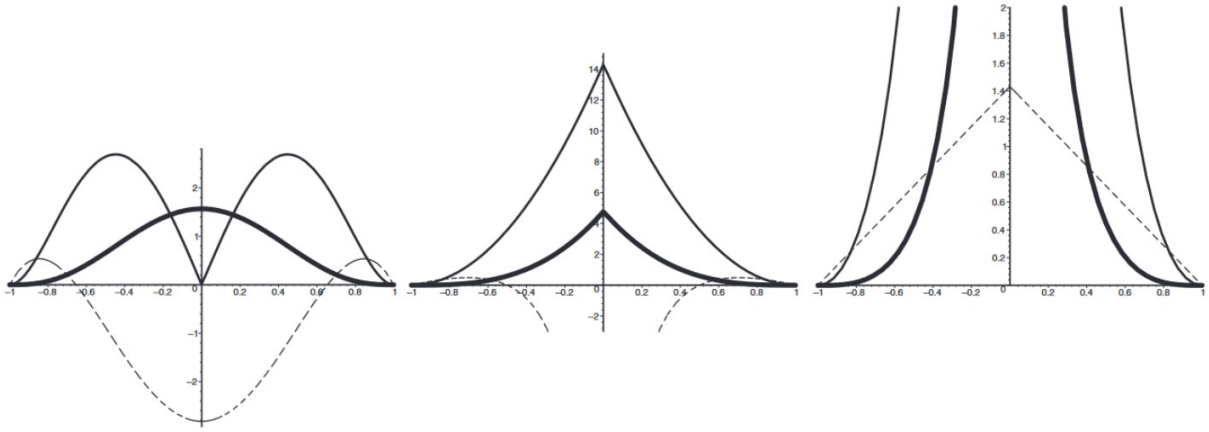


Figura 3.5: Representação grafíca dos *smoothing kernels: poly6*, *spiky e viscosity* respetivamente da esquerda para a direita. As linhas mais grossas mostram os *kernels*, as finas o seus gradientes e por fim, as linhas interrompidas representam os seus Laplacianos [34].

4

Simulação de Fluidos com Interatividade

O objetivo na criação deste ambiente de simulação de fluidos, para além de ter em vista o principal objetivo das *Computational Fluid Dynamics* (CFD), de reproduzir a descrição matemática da física de fluidos em um conjunto de instruções a serem executadas por um computador de maneira a representar o comportamento de um fluido, passa também por responder às questões de investigação citadas na secção 1.2. Com isto em consideração, o foco principal esteve em obter resultados plausíveis, mantendo a simplicidade, velocidade e precisão, podendo para isso sacrificar-se o aspeto visual.

Neste capítulo, o processo de desenvolvimento será descrito em maior detalhe, começando pela secção 4.1 onde é realizada a apresentação das diferentes otimizações de simulação implementadas, juntamente com o algoritmo de simulação de fluidos, utilizado como base de implementação. Ainda neste capítulo, é descrito o processo de desenvolvimento de um ambiente de simulação, tal como a sua finalidade. Para finalizar, são evidenciados os diversos controladores necessários para a apresentação do fluido desejado, seguidos dos diversos cenários, cruciais para uma correta e completa visualização dos possíveis comportamentos do fluido na secção 4.2.

4.1 Otimizações da Simulação

Até ao momento, foram explicados alguns conceitos teóricos fundamentais para a implementação do sistema de simulação de fluidos pretendido. Nesta secção, serão finalmente apresentadas as diversas técnicas e algoritmos utilizados neste projeto e de que forma estas foram implementadas com o objetivo de serem computadas pelo Unity.

Numa fase inicial do desenvolvimento, começou-se por fazer todo o processo de simulação de um fluido, do princípio, o que originou algumas dificuldades a obter resultados que parecessem satisfatórios e acima de tudo originou alguns problemas ao computar interceções do fluido com outros obstáculos. Na busca de soluções para resolver este problema, foi encontrado o código fonte de um projeto realizado por Leonardo Montes, que tinha como principal objetivo, criar uma simulação de vários personagens que interagiam

uns com os outros, como se de um fluido se tratasse. Ao analisar este projeto, entendeu-se que a sua abordagem era de certa forma semelhante à que tinha vindo a ser tomada na nossa implementação. Assim, foi tomada a decisão de utilizar este mesmo código fonte, como ponto de partida para a nossa implementação. A este primeiro algoritmo de simulação foi dado o nome de método *Base*, do qual se partiu para implementar algumas otimizações que melhorassem a sua eficiência. Para isso, foram propostos três algoritmos de otimização, simetria de forças (*Simetria de forças*), Monte-Carlo (*Monte Carlo*) e por fim, um método Híbrido do SPH (*Hash Grid*), que serão explicados em maior detalhe nas secções que se seguem.

Tendo em mente que se pretende criar um sistema de simulação de fluidos de Lagrange, o que significa que se usará uma abordagem baseada em partículas, é importante que primeiro se definam algumas estruturas de dados, que serão usados de forma geral, para todas as abordagens que serão tomadas na implementação deste sistema.

Estruturas de dados para um sistema de partículas: Ao implementar um simulador de físicas, baseado em partículas, é crucial que para isso sejam definidas determinadas estruturas de dados. O elemento-chave para a criação de um sistema de partículas é, por razões obvias, a própria partícula, e esta partícula pode ser constituída por vários componentes. Para isso foi criada a estrutura (*Struct*) 4.1 da qual será criado um vetor com o número de partículas desejado:

Por sua vez, sendo que é necessário distinguir quais os objetos da cena que serão responsáveis por interpretar o papel de obstáculos, é necessário criar também uma estrutura de dados para estes objetos. Tal estrutura pode ser observada na Listagem 4.2.

```
1 Struct SPHParticle {
2     Vector3 position;
3     Vector3 velocity;
4     Vector3 forcePhysics;
5     float density;
6     float pressure;
7     int parameterID;
8     GameObject go; }
```

Listagem 4.1: Estrutura de dados de uma partícula.

```
1 Struct SPHCollider {
2     Vector3 position;
3     Vector3 right;
4     Vector3 up;
5     Vector3 forward;
6     Vector3 scale;
7 }
```

Listagem 4.2: Estrutura de dados de um *collider*.

```
1  void InitSPH {
2      particles ← vector[amountOfParticles] of SPHParticle;
3      Random.seed;
4      forall particles i do {
5          float jitter ← float of random range;
6          float x ← float of SomePosition + (i % rowSize) + Random.range(-0.1f
       ,0.1f);
7          float y ← float of SomePosition + 2 + (float)(i / rowSize) * 1.1f;
8          float z ← float of SomePosition + ((i / rowSize) % rowSize + Random.
       range(-0.1f,0.1f);
9
10         go ← Instanciate GameObject:
11         go.transform.scale ← Vector3.One * particleRadius;
12         go.transform.position ← Vector3(x + jitter, y, z + jitter);
13         particles_i.Init(new Vector3(x, y, z), parameterID, go);
14         countReadyParticles++;
15     }
16 }
```

Listagem 4.3: Algoritmo de instanciação de partículas.

4.1.1 Base

Primeiramente, tal como enunciado na secção 4.1, será explicado em maior detalhe o funcionamento do algoritmo usado como base para o desenvolvimento das restantes otimizações. Para isso, ao analisar o ciclo de processamento do algoritmo, da Figura 3.3, é possível retirar uma primeira impressão de como este deve funcionar.

Inicialização: Dando inicio à sua implementação, é natural que se comesse pela iniciação/instanciação das partículas do fluido que se pretende simular, assim, no algoritmo 4.3 são então criadas as várias partículas apresentadas num formato cúbico, com lado de tamanho *rowSize*, que tenha sido introduzido pelo utilizador, no ficheiro *Data*.

Feita a inicialização das partículas, é necessário implementar todo o comportamento e físicas do fluido pretendido, ou seja, fazer uma implementação do SPH tal como descrito na secção 3.5.1. Para chegar à nova posição e velocidade da partícula, é preciso que primeiro se calcule a força aplicada nessa mesma partícula (resultado da equação 3.3). Esta equação, por sua vez, é decomposta por três componentes de força, força de pressão, viscosidade e das forças externas ao fluido. Por fim, para chegar ao resultado destas forças, é necessário calcular a pressão e densidade das partículas, no momento da iteração atual.

Densidade e pressão de cada partícula: Com o objetivo de chegar à força total que atua sobre as partículas, começa-se por chegar ao valor de pressão e densidade das mesmas. Para isso, utilizou-se o algoritmo 4.4, que fica responsável por calcular estas componentes para cada uma das partículas.

De notar, pela aplicação da linha dez do algoritmo 4.4, que da maneira que foi apresentado durante o capítulo 4 apenas as partículas que se encontram a uma distância menor

que o *Smoothing radius* (raio de afetação), serão, efetivamente, usadas para estes cálculos, menosprezando as partículas que não chegam a afetar a partícula atual.

```
1  void computeDensityPressure {
2      forall particles i do {
3          density_i ← 0.0f;
4          forall particles j do {
5              Vector3 rij;
6              float r2;
7              rij and r2 ← distBetweenParticles(i,j,out rij,out r2);
8
9              if r2 < smoothingRadiusSquared  than
10                 density_i ← density_i + particleDensity(r2);
11         }
12         pressure_i ← particlePressure(i);
13     }
14 }
```

Listagem 4.4: Algoritmo que calcula a densidade e a pressão das partículas.

Cálculo das forças aplicadas em cada partícula: Obtendo-se os valores de densidade e pressão de cada partícula, são cumpridos todos os requisitos para passar para o próximo passo, e assim, chegar ao resultado da aplicação da equação 3.3. Por outras palavras, é agora possível calcular as forças de pressão, viscosidade e da gravidade, do fluido a simular. Estes cálculos são efetuados pelo algoritmo 4.5, onde todas as interações de cada partícula são testadas e, caso estas entrem em interação, as forças de afetação de cada partícula são somadas, obtendo-se finalmente, a força da pressão, viscosidade e gravidade de cada uma. Por fim, ao somar todas estas forças, obtemos a força aplicada em cada partícula, na iteração atual.

```
1  void computeForces {
2      forall particles i do {
3          Vector3 forcePressure ← Vector3 of zeros;
4          Vector3 forceViscosity ← Vector3 of zeros;
5          forall particles j do {
6              if i is equals to j than
7                  Skip iteration of j;
8              Vector3 rij; float r2 float;
9              distBetweenParticles(i,j,out rij,out r2);
10             float r ← square of r2;
11             if r < smoothingRadius  than {
12                 forcePressure ← forcePressure + pressureForce(i,j,rij,r)
13                 forceViscosity ← forceViscosity + viscosityForce(i,j,r);
14             }
15         }
16         particles_i.forcePhysic←forcePressure + forceViscosity + gravityForce(i);
17     }
18 }
```

Listagem 4.5: Algoritmo que calcula as forças da densidade, pressão e gravidade a atuar em cada partícula.

```
1 void Integrate() {
2     forall particles i do {
3         velocity_i ← velocity_i + DT * (particles_i.forcePhysic) / density_i;
4         position_i ← position_i + DT * (velocity_i);
5     }
6 }
```

Listagem 4.6: Método que calcula a posição das partículas com integração do tempo.

Integração do tempo: Neste momento, possuímos a força total a agir sobre cada partícula. De forma a representar as partículas no cenário de simulação, é necessário que se aplique integração do tempo no seu movimento e calcular a posição e velocidade final da partícula. Para este fim, como sugerido por Simon Green [36] é utilizado o método de integração de Euler, que recorre ao uso da segunda lei de Newton para chegar às posições das partículas, onde a velocidade é atualizada com base nas forças aplicadas na partícula. De seguida, a posição é também atualizada, mas com base na velocidade calculada anteriormente. A implementação deste método pode ser analisada no código da Listagem 4.6.

Ao recordar o que foi anteriormente explicado na secção 3.5.1 e por Pierre Thuillier et al. [14], é possível concluir que, através da variação do momentum, é possível calcular a aceleração de uma partícula através da seguinte equação 4.1:

$$a_i = \frac{F_i}{\rho_i}, \tag{4.1}$$

Por sua vez, com recurso a um pequeno tempo de integração, permite computar a velocidade e a nova posição da partícula através das equações 4.2 e 4.3 que se seguem :

$$u_i = u_i + \Delta t \frac{F_i}{\rho_i}, \tag{4.2}$$

$$x_i = x_i + \Delta t u_i, \tag{4.3}$$

Computação das colisões: Após terem sido computadas todas as forças de interação entre as partículas do fluido e a força da gravidade, a posição prevista das partículas computadas pela integração do tempo pode interpor-se às condições impostas pelo cenário em que o fluido se encontra, isto é, a posição prevista para as partículas pode transpor possíveis obstáculos que partilhem cena com o fluido. Para corrigir este possível problema, o método apresentado na Listagem 4.7, é responsável por simular nas partículas, a velocidade que se espera que fosse causada pelo impacto das partículas com o obstáculo e da mesma forma, corrigir a previsão para a posição final das partículas.

```
1  void ComputeColliders() {
2      vector collidersGO ← findColliders;
3      vector SPHCollider.colliders ← new SPH.SPHCollider[collidersGO.Length];
4      forall colliders i do
5          Initialise colliders colliders_i ← collidersGO_i.transform;
6      forall particles i do {
7          forall colliders j do {
8              // Check collision
9              Vector3 penetrationNormal;
10             Vector3 penetrationPosition;
11             float penetrationLength;
12             if (Intersect(colliders_j, position_i, particleRadius, out
       penetrationNormal, out penetrationPosition, out penetrationLength)) than
13             {
14                 velocity_i ← DampVelocity(colliders_j, velocity_i, penetrationNormal,
       1.0f - particleDrag);
15                 position_i ← penetrationPosition - penetrationNormal * Mathf.Abs
       (penetrationLength);
16             }
17         }
18     }
19 }
```

Listagem 4.7: Método que calcula a posição e velocidade das partículas ao interagir com um obstáculo.

Aplicação das posições: Após todos os cálculos anteriores, fica apenas a faltar a atribuição dos resultados obtidos nos reais objetos de jogo, que estão a simular as partículas e o seu comportamento. Para isso, basta igualar as duas posições, tal como é apresentado no método que se segue na Listagem 4.8:

4.1.2 Simetria de Forças

Perante a necessidade de otimizar a frequência de atualização de imagem (FPS) obtida com o algoritmo de simulação inicial, o primeiro método de otimização que se idealizou passou por considerar que a densidade, pressão e forças aplicadas por uma partícula i em j, seriam por sua vez, simétricas às causadas por j em i, reduzindo assim as iterações necessárias, ou seja, o grau de complexidade do algoritmo, de $\mathbf{n}^2$ para apenas $\mathbf{n} * \mathbf{log_2}\,\mathbf{n}$ onde a variável **n** reflete o número de partículas constituintes do fluido.

```
1  void ApplyPosition() {
2      forall particles i do {
3              if particles_i.go != null than
4                  particles_i.go.transform.position ← position_i;
5          }
6  }
```

Listagem 4.8: Método que aplica as posições calculadas das partículas às reais partículas a simular.

Em virtude do aspeto mencionado no parágrafo anterior, começou-se por analisar cuidadosamente as equações apresentadas em 3.4 e 3.5. Contrariamente ao que se esperava, não foi dificil concluir que tal simetricidade não era, de facto, garantida. Tal condição, pode ser facilmente compreendida numa situação em que se esteja perante duas partículas que interagem entre si, uma vez que a densidade da partícula depende apenas da massa e da distância entre partículas, e o gradiente do kernel, depende principalmente dessa mesma distância e do *Smoothing radius*, que são iguais para ambas as partículas em interação. A partícula i fica apenas dependente da pressão da partícula j para assim calcular a sua força de pressão e vice versa. Uma vez que, por norma, a pressão na localização de cada partícula será diferente, é possível concluir que as forças de pressão não serão simétricas entre partículas em interação. Da mesma forma, também a força da viscosidade será assimétrica, uma vez que esta depende do campo de velocidade de cada partícula, sendo razoável esperar que divirja de partícula para partícula.

Para resolver este problema, são propostas duas alternativas, que por sua vez privilegiam a precisão dos resultados ou a velocidade de computação. O que se sugere é que ao invés de iterar a interação de todas as partículas com todas as restantes, tal como é feito no algoritmo da secção 4.1.1, esta interação é iterada em apenas um sentido, onde se calcula de uma vez, os valores de i e j na mesma iteração. Assim, o que difere nas duas alternativas, é justamente o modo como estes cálculos são efetuados. Para a nossa primeira proposta, apesar das partículas serem iteradas em apenas um sentido, o cálculo da pressão e viscosidade são realizados duas vezes, uma para a partícula i e outra para a j. O que acontece nesta abordagem é que para privilegiar alguma precisão dos resultados, a velocidade de execução é ligeiramente sacrificada, já que se continua a executar os cálculos para ambas as partículas em interação. Posto isto, e do mesmo modo que foi enunciado anteriormente, estes cálculos entre partículas não são propriamente simétricos, e é com base nesta situação que surge a nossa segunda proposta.

Ao analisar a nossa primeira alternativa, rapidamente se deu conta que, apesar da redução de iterações entre partículas contribuir imenso para um aumento da velocidade de computação, a necessidade de realizar os cálculos para ambas as partículas, partícula i com j e j com i, se tratava de uma questão que limitava a nossa otimização. Desta limitação surge a nossa segunda proposta, onde Müller et al. [34], propõe, apesar de já existirem diversas propostas na literatura, uma simples solução que observou ser a mais adequada para propósitos de maximização de velocidade e estabilidade, propósitos estes que se apresentam como as principais preocupações no estudo e desenvolvimento da presente dissertação. Assim, Müller, cria uma adaptação da equação 3.4, em que recorre à média das pressões das partículas em interação, resultando na equação 4.4:

$$f_i^{pressão} = -\nabla p = -\sum_j m_j \frac{Pi + Pj}{2\rho j} \nabla W(r_i - r_j, h) \tag{4.4}$$

De seguida, como forma de adaptação da equação 3.5, contando que as forças de viscosidade dependem apenas da variação de velocidade invés de velocidades absolutas,

o método de simetria mais natural será, de facto, recorrer à diferença de velocidades, tal como é possível observar pela equação 4.5:

$$f_i^{viscosidade} = v\nabla^2\vec{u} = v\sum_j m_j \frac{u_j - u_i}{\rho j}\nabla^2 W(r_i - r_j, h) \tag{4.5}$$

4.1.3 Monte-Carlo

Mantendo-se a concentração em reduzir o tempo de execução de cada atualização do sistema, resultando numa otimização da frequência de atualização de imagem (FPS), optou-se pela implementação de uma adaptação dos algoritmos de Monte-Carlo. Assim, como apontado anteriormente, métodos de Monte-Carlo (MC) têm vindo a ser aplicados numa vasta diversidade de problemas, tais como: inferências estatísticas, simulação e ou integração. Para além destes setores, métodos de Monte-Carlo, ainda têm vindo a ser utilizados com grande extensividade na computação gráfica, na teoria do transporte de luz baseado em físicas. Motivados por esta crescente utilização, Damien Rioux-Lavoie et al. [40], realiza a adaptação e demonstração da utilidade dos métodos de Monte-Carlo em *Computational Fluid Dynamics* (CFD). Desta forma, com a forte inspiração originada pelas diversas adaptações e implementações deste modelo de métodos, pretendeu-se adaptar a sua utilização para a seleção das interações entre partículas e analisar quais os resultados obtidos da sua aplicação.

A intenção que fundamenta a implementação de uma adaptação de Monte-Carlo, não consta em implementar uma simulação diferente da original, mas uma que mantenha os princípios da simulação por SPH seguidos até ao momento, visando apenas otimizar os cálculos realizados entre as partículas, que são uma das principais e mais demoradas operações na simulação do fluido.

Por definição, os métodos de Monte-Carlo, são orientados por estatísticas que se baseiam em amostras aleatórias, que originam resultados para problemas, que por princípio, podem ser considerados como sendo determinísticos. Como anteriormente enunciado, este tipo de métodos apresenta grande distinção na solução de problemas de otimização e integração numérica, apresentando grande utilidade para sistemas de simulação que disponham de diversos graus de liberdade[1], tais como as simulações de fluidos.

Apesar de existirem diversas abordagens a métodos de Monte-Carlo, estes tendem a seguir um padrão de desenvolvimento, começando por se definir o âmbito de *inputs* possíveis, implementar a distribuição probabilística sobre o domínio, realizar a computação determinística dos *inputs*, e por fim, agregar os resultados obtidos. Na nossa implementação, o âmbito inicial é definido pelas diferentes propriedades do fluido pretendido, tal como o ambiente em que este será simulado. De seguida, é necessário criar um valor aleatório para definir a função responsável pela distribuição de partículas a computar.

[1] O número de parâmetros independentes que definem a sua configuração e ou estado!

No Unity o *Ramdom* trata-se de um gerador numérico que produz uma sequência predefinida ao valor gerado. Com o objetivo de gerar resultados determinísticos sempre que a simulação é executada, para que seja possível realizar a análise dos resultados obtidos, é selecionada uma semente (*seed*) que a cada execução é responsável por gerar a mesma sequência de valores pseudoaleatórios.

Com o objetivo de otimizar a nossa implementação de Monte-Carlo de modo a atingir o melhor nível de desempenho possível, decidiu-se que esta deveria ser implementada utilizando a nossa anterior otimização, secção 4.1.2, como base para o seu desenvolvimento. Desta forma, a otimização de Monte-Carlo dispõe também das mesmas duas abordagens, apresentadas na secção 4.1.2. Contudo, o modo como a otimização de Monte-Carlo, itera entre as possíveis interações de partículas é ligeiramente diferente. Invés de iterar as listas de partículas, de todas com todas, dependendo do valor aleatório gerado na Listagem 4.9 pela percentagem desejada, cada partícula interage apenas com as ademais partículas que respeitam esse intervalo, passando apenas a ser considerada a percentagem de interações que o utilizador desejar.

Com esta otimização, para além de permitir que o utilizador escolha a percentagem de partículas em interação, ainda podemos ganhar rendimento ao selecionar percentagens de interação mais reduzidas. Em contrapartida, ao reduzir a percentagem de interações estaremos também a afetar a precisão dos resultados obtidos, podendo mesmo fazer com que a simulação deixe de se comportar como um fluido.

```
1    rnd = Random.Range(1, (amount / (int)(amount * particlePercentage)) * 2);
```

Listagem 4.9: Cálculo do valor aleatório responsável por definir a distribuição probabilístico.

4.1.4 SPH Híbrido

Até ao momento, os algoritmos de simulação de fluidos apresentados têm sido meramente guiados com base no ponto de vista de Lagrange, o que significa, que se baseiam apenas em partículas e nas suas componentes. Voltando-se um pouco atrás, relembrando a secção 2.2, é possível retirar que em relação à simulação de fluidos, existem dois importantes pontos de vista em relação a estas simulações. Sejam eles simulações baseadas em partículas e/ou em grelhas, sendo que cada uma delas apresenta certas vantagens na sua utilização. Ao analisar o livro [7], podemos mais uma vez retirar essa conclusão, onde métodos guiados por partículas, tal como o SPH, para além de serem mais simples de computar, ainda mantêm a conservação da massa e momentum de forma muito mais eficiente que os métodos guiados por grelhas. Em compensação, simulações baseadas em grelhas tendem a obter resultados muito mais suaves e precisos. Perante estas informações, não é fora do normal, começar a imaginar, juntar o melhor que cada um dos métodos tem para oferecer e criar um algoritmo de simulação de forma híbrida.

Ao analisar os algoritmos anteriores, é possível reparar que para usar métodos baseados em partículas, como o SPH, é necessário verificar todas as possíveis interações entre partículas. Considerando que estes algoritmos serão computados em apenas um *thread*, ou seja, sem paralelização, a operação de iterar todas estas combinações pode ficar bastante penosa, mesmo com as otimizações anteriores. Com este problema em mente, foi decidido que uma otimização alternativa, que pudesse aumentar substancialmente a velocidade da simulação, seria um algoritmo em que se recorresse a uma grelha auxiliar, com o objetivo de mapear com que partículas cada partícula interage e assim, evitar percorrer todas as possíveis partículas da simulação. Para isto, como sugerido por Simon Green [36] e utilizado em investigações como [20], [23] e entre diversas outras, optou-se por utilizar uma *Spatial uniform hash grid*.

4.1.4.1 Spatial Hashing e Uniform Grid

Uniform grid é uma técnica de divisão de espaço que subdivide o espaço de simulação em diversas células cúbicas do mesmo tamanho uniforme, formando assim uma grelha. Por sua vez, a escolha de recorrer a *spatial hash grids* partiu da investigação realizada [38], fundamentada graças à necessidade que aplicações em tempo real, como os video jogos, têm de simular o mais rapidamente possível as maiores e mais complexas simulações que sejam necessárias. Como expresso em [38], *spatial hashing* é uma das mais conhecidas técnicas, quanto à aceleração de *query operations*, neste caso, para a procura de partículas que sejam vizinhas umas das outras, criando assim uma interação entre as mesmas.

Spatial hashing, é uma técnica que mapeia as posições de objetos, estejam eles representados num ambiente 2D ou 3D, numa *hash table* em uma dimenção (1D) com recurso a uma *hash function*, que divide o ambiente de simulação em um conjunto de células, formando assim a citada *spatial grid*. Para isto, é necessário que o utilizador introduza algumas propriedades da grelha pretendida, como o tamanho e o número de partículas que podem ocupar a mesma célula. Analisar as Figuras 4.5 e 4.6.

Para além destas propriedades, também de extrema importância é a seleção do tamanho de cada célula da grelha. Por sua vez, este tamanho, influencia o número de objetos que são mapeados no mesmo índice de *hash*. Como forma de simplificar e com base no estudo realizado em [47] o valor decidido para o tamanho de cada célula é feito com base no diâmetro de cada partícula, ou seja, duas vezes o seu raio, pois, segundo o estudo realizado, se o tamanho das células for maior que o de uma partícula, o número de partículas por índice de *hash* também aumenta, abrandando assim os testes de interceção das partículas. Por outro lado, se o tamanho das células for menor que o de uma partícula, as partículas passariam a ocupar um maior número de células, aumentando assim o número de verificações necessárias para chegar a todas as interações das partículas.

Relativamente à mencionada *hash function*, esta foi implementada com base no livro de Doyub Kim [26]. Neste livro é mencionado que uma *hash function* que mapeie uma partícula especifica num valor chave (inteiro) pode ser feita de diversas maneiras, mas

```
1 Vector3Int GetCell(Vector3 position) {
2     return new Vector3Int((int)(position.x / CellSize), (int)(position.y /
      CellSize), (int)(position.z / CellSize));
3 }
```

Listagem 4.10: Método que converte a posição da partícula para uma posição da grelha.

```
1 int Hash(Vector3Int cell) {
2     return cell.x + DimensionX * (cell.y + DimensionY * cell.z);
3 }
```

Listagem 4.11: Método que converte as coordenadas 3D para um índice 1D.

que a melhor abordagem deve manter o mapeamento de forma espacial (*Spatial mapping*). Com esta finalidade a implementação realizada é feita com base nas seguintes funções 4.10 e 4.11:

As funções 4.10 e 4.11 são responsáveis por, em primeiro lugar, através da equação 4.10 receber a posição da partícula no mundo e converter a uma coordenada inteira que corresponda a uma célula de grelha. Seguidamente, a função 4.11 simplesmente mapeia as coordenadas inteiras 3D para um índice em uma dimensão, simulando a grelha num vetor linear.

4.1.4.2 Procura de Vizinhos

Ao implementar simulações de fluidos, baseadas em partículas, tal como é possível constatar ao analisar as otimizações anteriores, uma das operações mais executadas e que pode causar mais ou menos impacto na redução da velocidade de execução de um algoritmo de simulação de fluidos é, de facto, a procura das partículas que se encontram a uma distância de interação entre si. Assim, para que seja possível atingir simulações de fluidos com interatividade e que sejam eficientes, é necessário que o algoritmo de procura das partículas vizinhas seja também eficiente. Para além disso, ainda se deve ter a atenção de criar um algoritmo que seja paralelizável para que futuramente, caso seja pretendido, se implemente a simulação de forma paralela.

Criada a ***uniform grid*** com recurso a ***spatial hashing*** e terminado o mapeamento das partículas em células da grelha, através de uma tabela de hash, são criadas diversas divisões/células no cenário de simulação, a que Doyub Kim [26] identifica como ***buckets***. Assim sendo, a nossa simulação implementada em 3D, a partícula em foco apenas pode estar sobreposta num máximo de 8 *buckets*. Desta forma, ao realizar a busca pelas partículas que interagem com a partícula em foco, apenas é necessário iterar cada um dos ***buckets*** que sobrepõem a mesma, e avaliar, das partículas pertencentes a estes ***buckets***, quais respeitam o *smoothing radius* definido.

Deste modo, para além da necessidade de criar um algoritmo que em cada *frame* limpasse a tabela de *hash*, calculasse e alocasse as partículas nas suas posições atuais e que verificasse todas as interações entre partículas, foi também necessário implementar

uma função que determinasse quais os *buckets* que se sobrepõem à partícula em foco. Para isso, foi implementada uma função 4.12, baseada na implementação de Doyub Kim [26], que verifica, exatamente, quais os 8 *buckets* que sobrepõem a partícula, e verificar onde é que a posição da partícula se posiciona dentro do *bucket* com relação ao centro da célula.

4.2 Implementação do Sistema

Utilizando as técnicas e ferramentas indicadas na secção 3.4, o sistema proposto no capítulo 3 foi implementado com ênfase nos requisitos de um sistema em que o utilizador é um possível investigador que pretende observar o comportamento das partículas, estando estas expostas a diferentes cenários de simulação. Para além disso, houve também a preocupação que este sistema fosse acessível para utilizadores de diferentes matérias (podendo não haver um total à vontade perante a utilização de um sistema informático) em que utilizar diferentes *softwares* e aplicações externas, pudesse ser demasiado penoso e ou impossível para o utilizador em questão, continuando a ser necessária uma compreensão básica das propriedades e funcionamento de fluidos.

Perante o desafio de criar um sistema de simulação de fluidos, é natural que um dos primeiros problemas encontrados, seja, como é que se vai criar um ambiente 3D em que seja possível criar vários cenários para a simulação, e traduzir todas as computações necessárias para representar a física e o comportamento de um fluido, graficamente, e em tempo real, de forma contínua. Sendo assim, mais uma vez, é justificada a utilização do motor de jogos Unity, que para além de já conter os atributos necessários para resolver estes problemas, também é capaz de executar todas as etapas e técnicas necessárias para a representação de uma simulação de fluidos num ambiente em 3D. Ademais, em conformidade com o concluído por R. Nóbrega et al. [35] estes motores fornecem também a capacidade de utilizar deversas técnicas de vizualização e interação com o ambiente criado.

Ambiente de simulação: Ao usar uma ferramenta como o Unity, o processo para a criação do ambiente de simulação ficou facilitado, podendo fazer-se uso de ferramentas como, *Lights*, *Materials* e claro diversas primitivas com as quais é possível criar qualquer volume de simulação pretendido. Na Figura 4.1 é possível observar todas estas propriedades e a área em que fica presente a simulação. Para criar os volumes de simulação, foram utilizadas as primitivas denominadas de *Quads*, um plano de dimensões customizáveis com a dimensão do eixo y, coordenada Z do *transform*, fixa a 1.

Já que o objetivo passa por não utilizar *Colliders e Rigidbodies* fornecidos pelo Unity, é necessária a criação da *Tag*, SPHCollider (Figura 4.2), associada aos *Quads*, para que durante a simulação seja possível computar as colisões com os planos que realmente fazem parte do volume em questão. Após a criação do volume de simulação, é também necessária, uma câmara e luzes para observar a simulação em tempo de execução.

```
1  int GetNearbyKeys(Vector3Int originIndex, Vector3 position) {
2      Vector3Int[] nearbyBucketIndices ← new Vector3Int[8];
3              forall  i  from 0...7  do//8 cause the eight octantes
4                  nearbyBucketIndices[i] ← originIndex;
5
6              if ((originIndex.x + 0.5f) * SpatialHashing.CellSize <= position.x) {
7                  nearbyBucketIndices[4].x += 1;
8                  nearbyBucketIndices[5].x += 1;
9                  nearbyBucketIndices[6].x += 1;
10                 nearbyBucketIndices[7].x += 1;
11             }
12             else{
13                 nearbyBucketIndices[4].x -= 1;
14                 nearbyBucketIndices[5].x -= 1;
15                 nearbyBucketIndices[6].x -= 1;
16                 nearbyBucketIndices[7].x -= 1;
17             }
18             if ((originIndex.y + 0.5f) * SpatialHashing.CellSize <= position.y) {
19                 nearbyBucketIndices[2].y += 1;
20                 nearbyBucketIndices[3].y += 1;
21                 nearbyBucketIndices[6].y += 1;
22                 nearbyBucketIndices[7].y += 1;
23             }
24             else{
25                 nearbyBucketIndices[2].y -= 1;
26                 nearbyBucketIndices[3].y -= 1;
27                 nearbyBucketIndices[6].y -= 1;
28                 nearbyBucketIndices[7].y -= 1;
29             }
30             if ((originIndex.z + 0.5f) * SpatialHashing.CellSize <= position.z) {
31                 nearbyBucketIndices[1].z += 1;
32                 nearbyBucketIndices[3].z += 1;
33                 nearbyBucketIndices[5].z += 1;
34                 nearbyBucketIndices[7].z += 1;
35             }
36             else{
37                 nearbyBucketIndices[1].z -= 1;
38                 nearbyBucketIndices[3].z -= 1;
39                 nearbyBucketIndices[5].z -= 1;
40                 nearbyBucketIndices[7].z -= 1;
41             }
42             int[] nearbyKeys ← new int[8];
43             forall  i  from 0...7  do
44                 nearbyKeys[i] ← SpatialHashing.Hash(nearbyBucketIndices[i]);
45
46             return nearbyKeys;
47         }
48 }
```

Listagem 4.12: Método que converte as coordenadas 3D para um índice 1D.

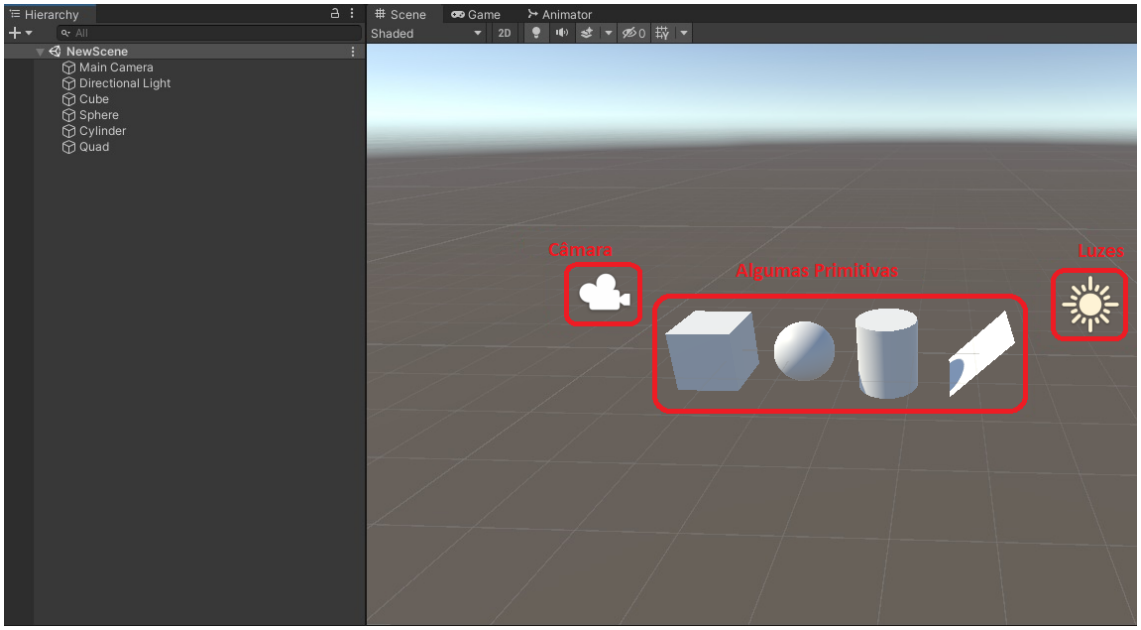


Figura 4.1: Figura de exemplo, onde é possível observar uma câmara, diversas primitivas e luzes, no motor de jogos Unity.

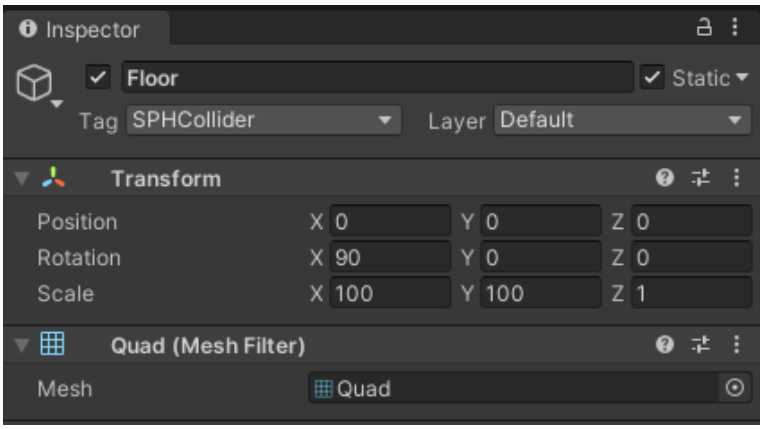


Figura 4.2: Figura onde é possível observar a associação da Tag SPHCollider, ao Quad "Floor".

Neste momento, tudo o que é necessário para um simples ambiente de simulação terá sido devidamente criado, ficando apenas a faltar a criação do fluido e claro, as computações necessárias para que este se comporte como um fluido real.

Controlador: Da lista de otimizações propostas na secção 4.1, foram, então, criados vários *scripts* responsáveis por criar as simulações desejadas com cada um dos algoritmos propostos, gerando uma vasta variedade de possibilidades, no que diz respeito à formação dos fluidos. Com a finalidade de facilitar esta seleção, é criado um *gameObject* e a este, adicionada uma nova componente, um controlador (*"Master Controller"*), responsável por estabelecer que tipo de cenário e ambiente de simulação se pretende (controlando assim, os vários algoritmos de otimização).

O *"Master Controller"* necessita que o utilizador preencha determinados requisitos, começando por selecionar de que forma pretende que as partículas sejam instanciadas, podendo estas ser, em grupo, onde todas serão criadas em formato de bloco e em "simultâneo", ou uma por uma, onde estas serão criadas em linha e uma de cada vez (esta opção foi criada com o principal objetivo de servir na recolha de dados estatísticos). De seguida, é necessária a seleção de um dos quatro métodos/algoritmos de simulação de fluidos anunciados na secção 4.1. Na eventualidade do utilizador pretender observar o algoritmo de otimização de Monte-Carlo, é ainda exposto à possibilidade de selecionar qual a percentagem de partículas a executar a interação. Para além disso, o utilizador ainda terá de selecionar em que ambiente de simulação pretende simular o fluido e de que forma pretende observar a interação entre as partículas. Estes cenários e modos de observação serão demonstrados, respetivamente, nas secções 4.2.1 e 4.2.2. Por fim, é ainda necessário, que o utilizador atribua ao controlador, a componente *Data* com as propriedades do fluido. Na Figura 4.3 é possível observar a disposição do *controller* citado.

Data: Como referido anteriormente, para simular o comportamento físico de um fluido, escolher o algoritmo e a técnica de simulação, não é suficiente sem que primeiro se saiba quais as propriedades que se pretende que o fluido apresente. Estes podem assumir todo o tipo de comportamento, como o mel e a água, em que claramente o primeiro apresenta um comportamento muito mais viscoso que o segundo, e é aqui que entra a necessidade de existirem diversas propriedades capazes de descrever este mesmo comportamento.

Para atribuir estas propriedades, o master controler recebe um *Data* com todas as informações, e fica responsável por atribuir às partículas e ao fluido em si, todas estas propriedades. Assim, para se obter a simulação de um fluido que apresente certo comportamento, é importante que o ficheiro *Data* seja preenchido com as devidas propriedades,

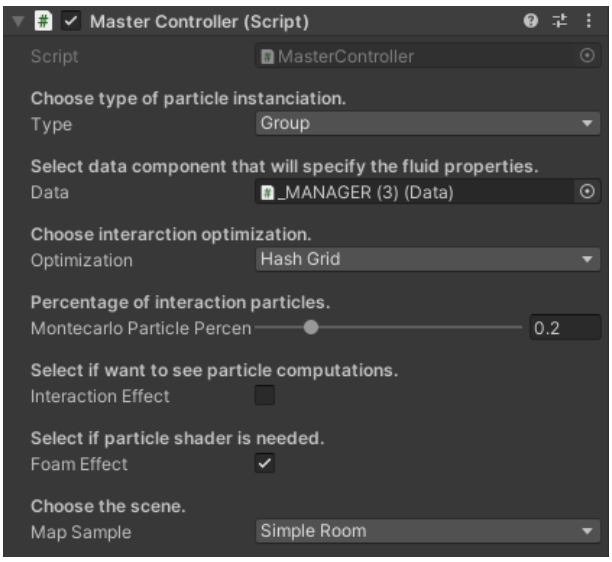


Figura 4.3: Figura onde é possível observar as várias componentes necessárias para o controlo e criação do fluido.

a fim de simular o fluido com as características pretendidas. Estas propriedades podem ser vistas na Figura 4.4 e posteriormente na tabela 4.1 é possível analisar e perceber cada uma das propriedades disponíveis.

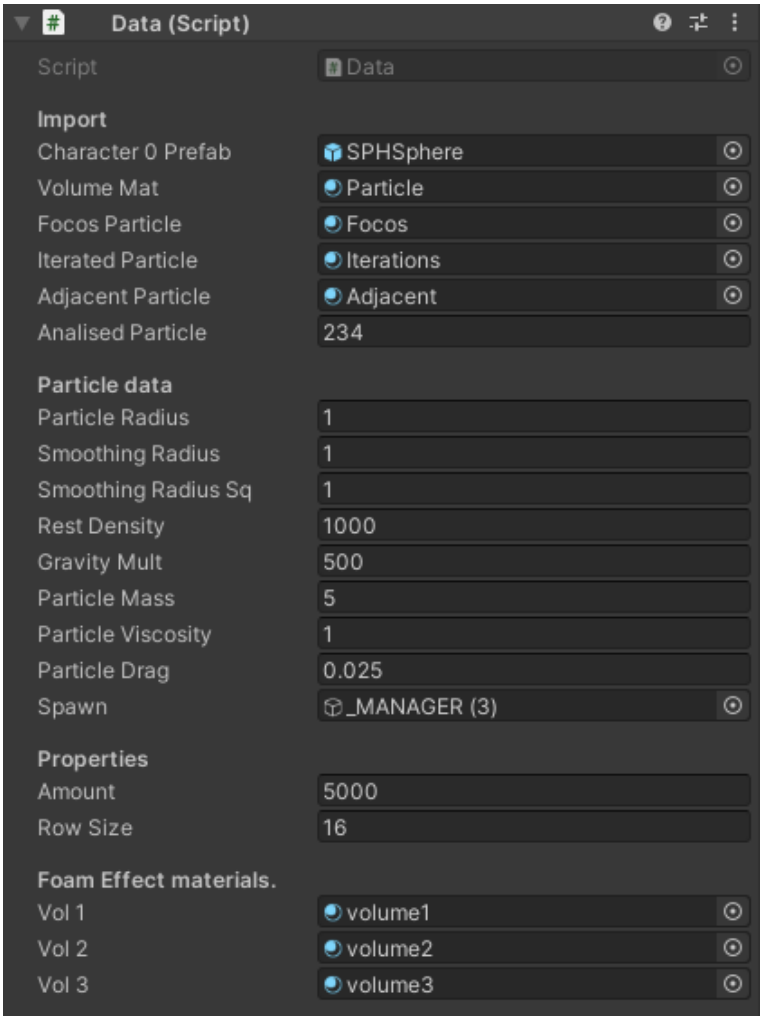


Figura 4.4: Figura onde é possível observar as várias propriedades necessárias para a simulação de um fluido.

Grelha de simulação: Na eventualidade do utilizador pretender criar uma simulação de fluidos com a otimização com recurso a uma *Hash Grid*, ou seja, uma técnica híbrida do SPH, é necessária mais uma componente de controlo do fluido. O utilizador precisa atribuir ao *gameObject* do controlador, um *script* do *hash grid* onde terá de colocar as dimensões da grelha, necessárias para simular o fluido em determinado volume no espaço, e quantas partículas podem ocupar cada célula desta grelha. Na Figura 4.5 é possível observar um exemplo para a construção deste controlador.

Tabela 4.1: Propriedades que definem o fluido

Nome da Propriedade	Conteúdo	Tipo
Character 0 Prefab	Material predefinido das partículas	GameObject
Volume Mat	Material das partículas de água	Material
Focos Particle	Material da partícula em foco	Material
Iterated Particle	Material das partículas iteradas pela partícula em foco	Material
Adjacent Particle	Material das partículas a interagir com a partícula em foco	Material
Analysed Particle	Partícula que se pretende usar como foco	Integer
Particle Radius	Raio da partícula	Float
Smoothing Radius	Raio de interação entre partículas	Float
Smoothing Radius sq	Raio de interação entre partículas ao quadrado	Float
Rest Density	Densidade em que uma pequena porção de fluido é considerada como estando em repouso	Float
Gravity Mult	Multiplicador de gravidade	Float
Particle Mass	Massa de cada partícula	Float
Particle Viscosity	Viscosidade do fluido pretendido	Float
Particle Drag	Força que atua no sentido oposto ao movimento das partículas que se movem em relação ao ambiente envolvente	Float
Spawn	Posição no espaço onde o fluido será instanciado	GameObject
Amount	Número de partículas que se pretende instanciar	Integer
Row size	Tamanho em quantidade de partículas de cada fileira do bloco	Integer
Vol1	Material das partículas que se movem com velocidades mais lentas	Material
Vol2	Material das partículas que se movem com velocidades médias	Material
Vol3	Material das partículas que se movem com velocidades mais rápidas	Material

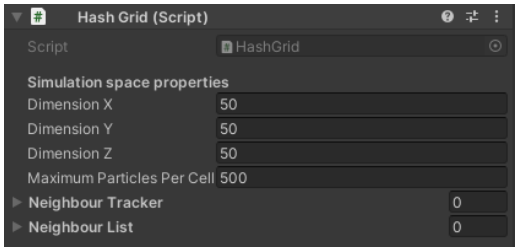


Figura 4.5: Figura onde é possível observar as várias propriedades necessárias para criação de uma grelha de simulação de fluidos.

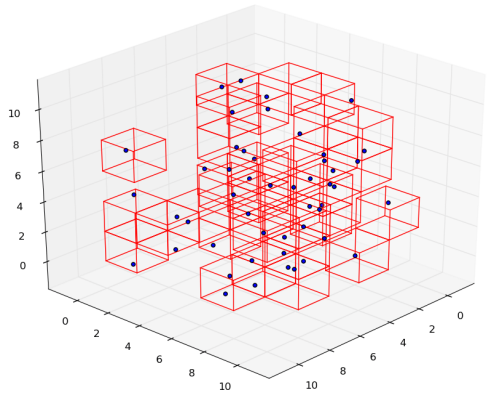


Figura 4.6: Exposição gráfica da Spatial hash grid onde as partículas são mapeadas em células de tamanho uniforme [13].

4.2.1 Cenários de Simulação

Como mencionado na secção inicial do capítulo, no decorrer do desenvolvimento do sistema de simulação de fluidos foram implementados diversos cenários de simulação dos quais o utilizador dispõe para visualizar diferentes simulações, como salpicos, efeitos de cascata, ondulação, entre outros possíveis efeitos. Nesta secção serão apresentados os diversos cenários de simulação implementados.

Caixa Simples: Numa fase inicial, ainda sem um algoritmo de simulação de fluidos que funcionasse corretamente, com o objetivo de criar um cenário onde fosse possível visualizar, com facilidade, o comportamento de todas as partículas e o impacto que a alteração de cada propriedade do fluido (que podem ser observadas na Figura 4.4) causaria nesse mesmo comportamento, optou-se por um volume de simulação mais genérico e simples, para que o foco estivesse apenas na observação do seu comportamento o mais natural possível, sem intervenção de outros objetos ou condições que o alterem. Neste cenário é possível observar a queda de um bloco de partículas, como se de uma gota de fluido se tratasse, e o comportamento da mesma ao colidir com o solo causando um efeito semelhante a um salpico. Este primeiro cenário pode ser observado na Figura 4.7.

Plano Inclinado (rampa): De seguida, com o objetivo de ter uma melhor perceção de como a viscosidade pode afetar o movimento do fluido enquanto escorre, foi criado um cenário composto por um plano inclinado que simula o mesmo comportamento que uma rampa causaria. A fim de tornar o cenário mais completo, surgiu a ideia de adicionar uma pequena caixa no final da rampa, para onde as partículas pudessem escorrer. Com isto, foi possível observar que ao chegar a essa caixa, o fluido teve um comportamento semelhante ao final de uma cascata, criando alguma ondulação, que no final origina um movimento em arco, semelhante a um tubo. Apesar de impressionante e para além do

interesse que este comportamento do fluido pode originar, este cenário ainda permite, ao utilizador, observar as diversas mudanças que cada propriedade do fluido pode gerar. Um bom exemplo que o demonstra é o aumento do nível de viscosidade das partículas, que altera completamente o comportamento do fluido simulado, fazendo com que este desça pelo plano inclinado muito mais lentamente. Este cenário pode ser observado na Figura 4.7.

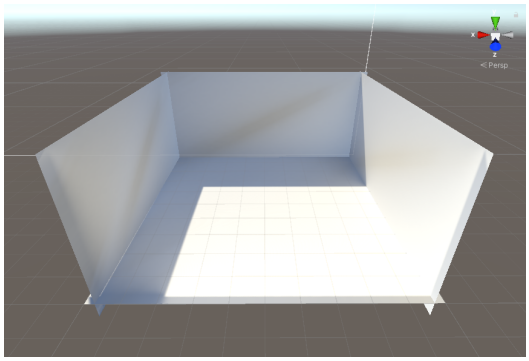
Câmara de Ondas: Este cenário foi criado com a finalidade de mostrar que é possível criar uma simulação de fluidos em tempo real, em Unity, e com interação, também computada em tempo real. Para isso, começou-se por criar uma caixa simples, semelhante à utilizada no primeiro cenário, com a adição de dois planos verticais colocados nas extremidades da caixa. Seguidamente, foram criadas duas animações em velocidades diferentes, que são responsáveis por movimentar estes planos e assim, ao interagirem com o fluido, são desse modo criadas as ondas pretendidas. Apesar de tudo aparentar estar a funcionar corretamente, esta interação é limitada pela taxa de atualização do sistema, pois se o tempo entre cada *update* for mais elevado e o corpo a interagir com o fluido se movimentar com demasiada velocidade, a interação não é computada com a rapidez suficiente para que seja aplicada, ficando difícil encontrar uma combinação perfeita entre a taxa de atualização do sistema e a velocidade dos obstáculos. Para dar início ao movimento dos planos verticais foi criada uma simples interface de utilizador, em que o mesmo pode clicar em dois botões e escolher quando iniciar e terminar o movimento de cada plano. O cenário apresentado pode ser observado na Figura 4.7.

Câmara com Obstáculos: Este cenário foi criado com a finalidade de mostrar a interação do fluido com os obstáculos presentes no cenário de simulação. Deste modo, foi criado um cenário fechado, inspirado na câmara de ondas, mas com um obstáculo a bloquear o livre movimento do fluido. O que se pretende com este cenário, é observar se o fluxo do fluido interage com o mesmo, e consequentemente, se continua o seu comportamento previsto, contornando o obstáculo e preencher o volume da câmara de ondas. O cenário apresentado pode ser observado na Figura 4.7.

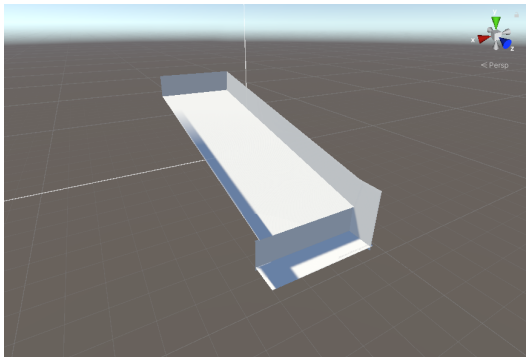
4.2.2 Modos de Observação

De forma a dispor de uma melhor visão da simulação e permitir que o utilizador retire o maior número de informações possível, com mais facilidade e eficácia, é fundamental que o utilizador seja capaz de alternar entre múltiplas câmaras, situadas em diferentes posições do cenário em que se encontra a simulação.

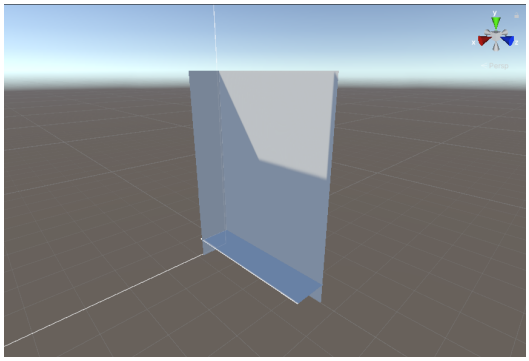
Outro requisito para o visualizador, relacionado com a interação e visualização do utilizador, especificado na secção 3.2, é a habilidade de escolher o modo de observação pretendido. Considerando que, o que é proposto no capítulo 3 é um sistema de simulação de fluidos que possa ser utilizado por investigadores, é importante que o sistema disponha



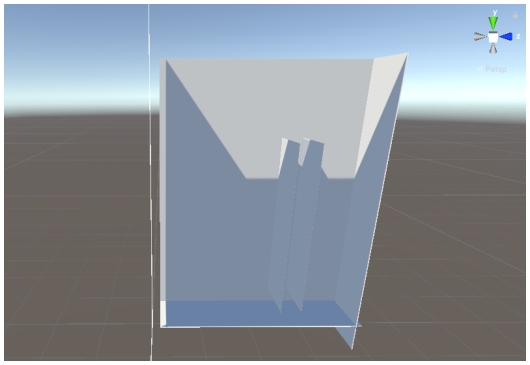
(a) Caixa Simples.



(b) Plano inclinado.



(c) Câmara de ondas.



(d) Câmara com obstáculos.

Figura 4.7: Diferentes cenários implementados no simulador.

de modos de observação que ofereçam ao utilizador uma melhor análise e compreensão do comportamento do fluido. Para oferecer ao utilizador essa capacidade de observar o comportamento do fluido com foco em diferentes características do mesmo, foram criados dois modos/efeitos de observação, como o efeito de interação e efeito de espuma, que serão especificados nos parágrafos seguintes.

Efeito de Interação: Para este primeiro efeito, o objetivo era oferecer ao utilizador uma funcionalidade que permitisse uma melhor análise da eficiência de cada algoritmo, podendo para isso, distinguir quais as partículas que estão a ser iteradas (testadas pelo método SPH) e quais estão, de facto, a influenciar o comportamento da partícula desejada. Para este efeito basta ativar a opção *"Interaction effect"* disponível no *Master Controller*, e escolher o número da partícula que se pretende analisar as interações, preenchendo a propriedade denominada de *Analysed Particle* no ficheiro *Data*.

Na Figura 4.8 é possível obter uma primeira impressão do funcionamento e utilidade deste efeito.

Efeito de Espuma: Neste efeito, a ideia era oferecer uma melhor impressão de quais as partículas que se movimentam mais rapidamente e quais as que se encontram praticamente estáticas, ou seja, com menos interações a acontecer. A utilização deste efeito é particularmente interessante perante o cenário "Câmara de ondas"apresentado na secção

anterior, pois ao infligir contacto e por sua vez uma nova força externa às partículas, é possível observar a sua movimentação em mais detalhe, incluindo a passagem de força entre as partículas onde foi aplicada a força, e as partículas que estão a uma distância de contacto com estas.

Na Figura 4.9 é mais uma vez possível observar o funcionamento deste efeito.

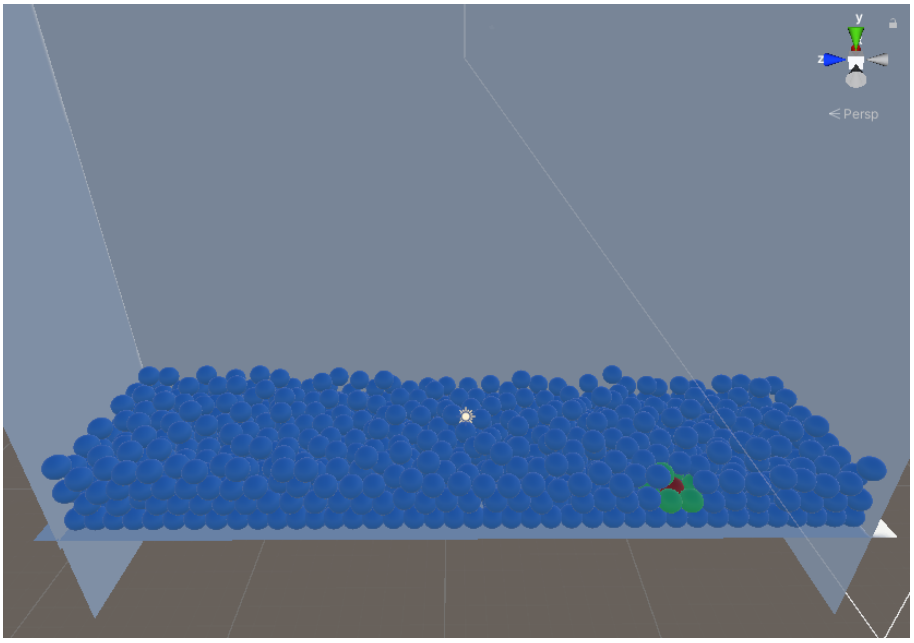


Figura 4.8: Figura onde é possível observar o efeito de interação entre as partículas, onde, a vermelho é representada a partícula em foco, e em verde, as que estão a interagir com o foco no momento.

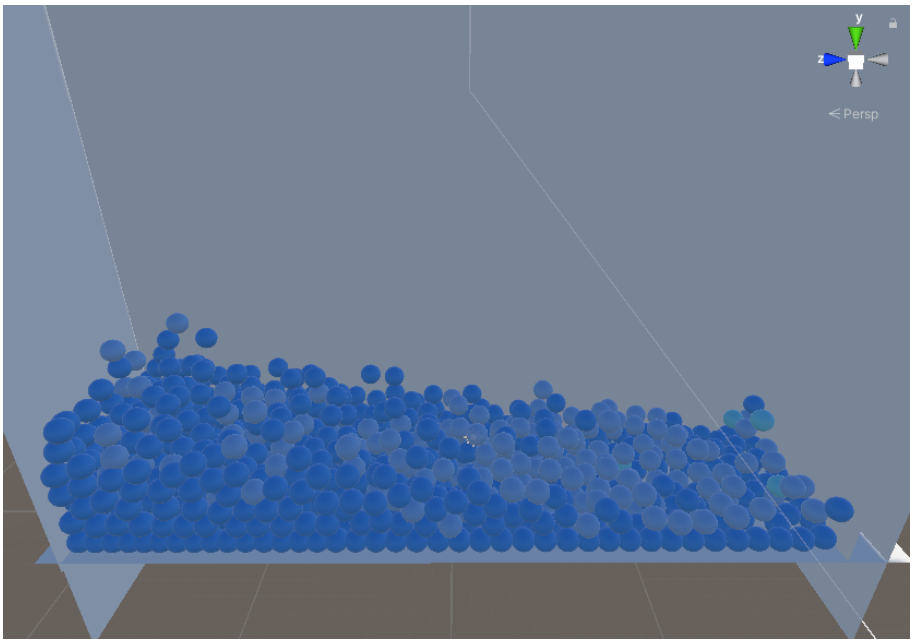


Figura 4.9: Figura onde é possível observar o efeito de espuma e onde é possível observar as partículas que se movimentam mais rapidamente, no momento.

5

Avaliação

De modo a avaliar os diferentes resultados dos algoritmos de simulação de fluidos implementados, definimos determinadas métricas cruciais para a comparação de desempenho e eficiência da nossa simulação, quando aplicadas com os diversos algoritmos apresentados. Em adição a estas estatísticas, e porque esta dissertação se foca na simulação dos comportamentos de um fluido líquido, é importante avaliar estes mesmos comportamentos. Assim, foram definidos alguns cenários de simulação de fluidos que, neste capítulo, serão apresentados e postos à prova quanto à sua correta interação com o ambiente envolvente, e se esse comportamento pode ser considerado convincente, no que diz respeito ao previsto comportamento de um fluido.

5.1 Estatísticas e Análise de Desempenho

Nesta secção são apresentadas as diversas métricas utilizadas para a avaliação do desempenho das diferentes otimizações implementadas. Com base nestas estatísticas, é realizada uma apreciação critica da funcionalidade de cada otimização e de quais as possíveis situações em que cada uma poderá desempenhar melhor os objetivos a que se propõe.

5.1.1 Especificação de Sistema

A implementação final desta dissertação consiste num simulador de fluidos, escrito na linguagem de programação C# e com recurso ao conhecido motor de jogos, Unity. O simulador criado é constituído por uma simulação de fluidos com base em *Smoothed-particle hydrodynamics*(SPH) e é capaz de simular uma grande variedade de fluidos líquidos com diferentes propriedades. O simulador disponibiliza ainda um algoritmo de simulação utilizado como base, complementado com três otimizações, tal como descrito nas capítulos anteriores. Por fim, estas simulações e otimizações serão, no decorrer deste capítulo, avaliadas e comparadas em termos de funcionalidade e velocidade de computação. Estas implementações e avaliações foram realizadas num computador com as especificações enunciadas na tabela 5.1.

Tabela 5.1: Especificações do computador utilizado durante a recolha de estatísticas.

Componente	Especificação
GPU	AMD Radeon RX 5700
CPU	AMD Ryzen 5 5600X 6-Core Processor 3.70 GHz
RAM	16,0 Gb
OS	Windows 11

5.1.2 Análise Geral de Otimizações

No processo de seleção das potenciais métricas utilizadas para a avaliação da nossa simulação de fluidos, a primeira estatística considerada foi a dos quadros por segundo, conhecidas como FPS, uma vez que se trata da estatística mais utilizada na avaliação da fluidez de um vídeo jogo. Contudo, rapidamente se chegou à conclusão que, apenas utilizar a métrica dos FPS não era, de forma alguma, suficientemente conclusivo na avaliação dos resultados obtidos, uma vez que, mesmo que se observe uma elevada taxa de FPS, a simulação, ou jogo, pode conter certas perdas de *frames*, causando as quebras na fluidez da imagem. Com isto em mente, propôs-se a adição de métricas como o número de partículas simuladas, o tempo decorrido durante a simulação e o tempo de computação de cada *frame*. Por fim, de forma a realizar uma análise mais detalhada das diferentes etapas da simulação, os tempos de simulação e instanciação do fluido, foram divididos em duas métricas. Na tabela 5.2, estão exibidas as diferentes métricas utilizadas nesta etapa de avaliação.

A primeira métrica, distinguida pelo identificador "M1", corresponde, como o nome indica, à quantidade de partículas a serem simuladas simultâneamente. Como anteriormente estabelecido no decorrer dos capítulos 3 e 4, pretende-se que o utilizador do nosso sistema tenha o total controlo sobre as propriedades do fluido que deseja simular, como tal, a capacidade de selecionar o número de partículas a simular representa um grande papel no resultado final do fluido, já que a quantidade de partículas não só influencia diretamente o volume do fluido como a sua densidade e desempenho. Assumindo que o aumento na quantidade de partículas a simular não terá um impacto linear nos cálculos de interação entre partículas e o comportamento do fluido, esta métrica é crucial para entender o real impacto de cada otimização, e de quais as otimizações mais adequadas para cada simulação que se pretenda.

A segunda métrica, de identificador "M2", foi o tempo de simulação decorrido. Esta

Tabela 5.2: Métricas de comparação de algoritmos

	Métrica
M1	Número de partículas
M2	Tempo de simulação decorrido
M3	Tempo de instanciação do fluido
M4	Tempo de computação de cada frame
M5	*Frames* por segundo

métrica representa o tempo total, desde o momento em que se inicia a simulação até a um momento previamente estabelecido. Esta métrica permite-nos analisar o progresso da nossa simulação e quanto tempo levou até que certos pontos fossem atingidos. Assim, de maneira a providenciar uma análise com maior precisão, esta métrica foi dividida em tempo de instanciação de partículas e o tempo de simulação do comportamento das mesmas. De notar que, sendo o principal foco das nossas otimizações, uma melhoria na busca pelas partículas em interação e consequentemente no cálculo das forças atuadas entre as mesmas, esta métrica permite precisamente a análise de ditas computações.

Em adição à métrica referida anteriormente, decidiu-se que destacar o tempo de intanciação, identificado por "M3", seria uma mais valia para uma análise mais cuidada e completa das nossas otimizações. Com esta métrica, para além de esclarecer quais as otimizações que causam um maior impacto na inicialização da simulação, ainda tem a finalidade de aprofundar a avaliação da nossa otimização com grelha auxiliar, uma vez que as dimensões desta grelha, são também totalmente parametrizáveis.

A métrica seguinte, de identificador "M4", refere-se ao tempo de computação de cada *frame*, correspondendo assim, ao tempo de computação das forças a atuar em cada partícula. Com esta métrica, pretende-se que a análise dos tempos de computação das referidas forças tenha um nivel de precisão superior, uma vez que se refere apenas ao tempo utilizado para referidos cálculos, excluindo tempos intermédios, desenho, *render* e restantes operações executadas pelo Unity.

Por fim, a métrica mais comum no universo dos vídeo jogos e simulações, o número de quadros por segundo (FPS), identificado por "M5", e com a finalidade de auxiliar na apreciação mais geral, do desempenho de cada otimização.

5.2 Análise de Desempenho entre Algoritmos

Nesta secção, é possível observar a aplicação das diferentes métricas enunciadas na secção anterior e com base nas estatísticas observadas, foi realizada a análise dos diferentes resultados obtidos. Com isto, foi possível avaliar o desempenho de cada algoritmo, de que forma se obtêm os resultados visualizados, e em que situações cada algoritmo poderá ou não ser privilegiado em relação aos ademais.

5.2.1 Comparação de Desempenho das Otimizações

De modo a obter os resultados adequados para uma avaliação válida entre os diversos algoritmos apresentados na tabela 5.3, estes foram colocados perante as mesmas circunstâncias. Nesta situação, concluiu-se que o mais adequado seria, que o cenário selecionado fosse o mais simples possível, sem a presença de quaisquer eventos, e que as suas propriedades de fluido fossem exatamente iguais. Deste modo, garantimos que nenhuma otimização terá algum tipo de privilégio demasiado expressivo em relação às restantes.

Tabela 5.3: Algoritmos de otimização

	Algoritmo de otimização
Algo.1	Algoritmo Padrão
Algo.2	Algoritmo de Simetria de forças
Algo.3	Algoritmo de Monte-Carlo
Algo.4	Algoritmo com adaptação híbrida

Tabela 5.4: Frequência de FPS correspondente a cada algoritmo para diferentes quantidades de partículas.

Nº Partículas	100	200	300	400	500	600	700	800	900	1000
FPS com Algo.1	431	238	128	79	53	38	28	22	17	14
FPS com Algo.2	502	353	210	137	95	69	53	41	33	27
FPS com Algo.3	528	482	334	240	179	136	108	87	71	60
FPS com Algo.4	437	444	372	314	278	244	214	195	176	159

Na tabela 5.4 é possível observar a frequência de FPS, atingida pela nossa simulação, quando composto por determinado número de partículas e aplicada num cenário como a "Caixa simples"apresentado na secção 4.2.1. Ao examinar esta tabela com menos detalhe, a primeira impressão que retiramos é que, de facto, o nosso Algo.4 (algoritmo com recurso a uma *hash grid* auxiliar) é o mais eficiente, uma vez que é capaz de simular muito mais partículas mantendo uma boa frequência de FPS. No entanto, ao realizar uma observação mais cuidada, este comportamento pode parecer no mínimo inusitado, apesar de quando exposto a uma maior quantidade de partículas, este parecer o mais adequado. No extremo oposto da nossa tabela, onde a quantidade de partículas a simular é muito inferior, verifica-se que o seu destaque em relação aos ademais algoritmos deixa de se manifestar. Com o objetivo de entender melhor este fenómeno, o dito algoritmo será explorado mais minuciosamente, na secção 5.2.5.

Apesar deste pequeno detalhe, na maioria dos casos, Algo.4, aparenta ser o mais eficiente, de tal maneira que, ao analisar os gráficos da Figura 5.1 e 5.2, esta tese, continua a ganhar ainda mais força ao concluir que para além de ser capaz de computar uma quantidade superior de partículas, ainda demonstra ser aquele que termina a simulação no intervalo de tempo mais reduzido.

Regressando ao declarado no principio da secção 5.1.2, a frequência de FPS pode muitas vezes induzir a erros de análise, uma vez que, apesar da simulação executar a 60FPS, não é garantido que cada *frame* dure os 16ms. Uma situação que o demonstra é, se 1 *frame* demorar 900ms e os restantes 59 *frames* apenas 1ms, a simulação continua a executar a 60FPS, mas não terá a fluidez que se espera. Assim, surge a necessidade de analisar as restantes métricas, apresentadas na tabela 5.2, para todos os algoritmos propostos 5.3.

No gráfico da Figura 5.2 é exibido o tempo decorrido, em milissegundos, desde o

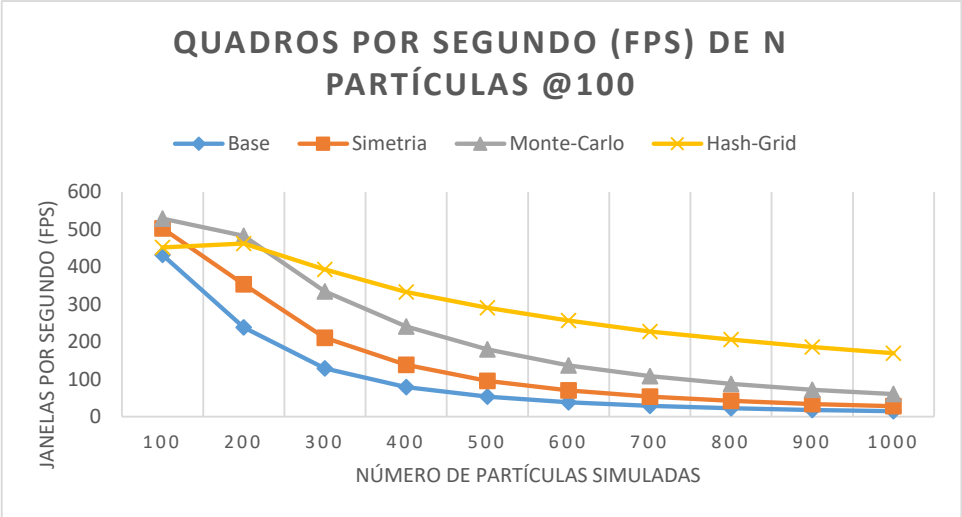


Figura 5.1: Número de quadros por segundo FPS a que cada algoritmo executa cada quantidade de partículas a simular nos primeiros 100 quadros, durante todas as etapas de simulação.

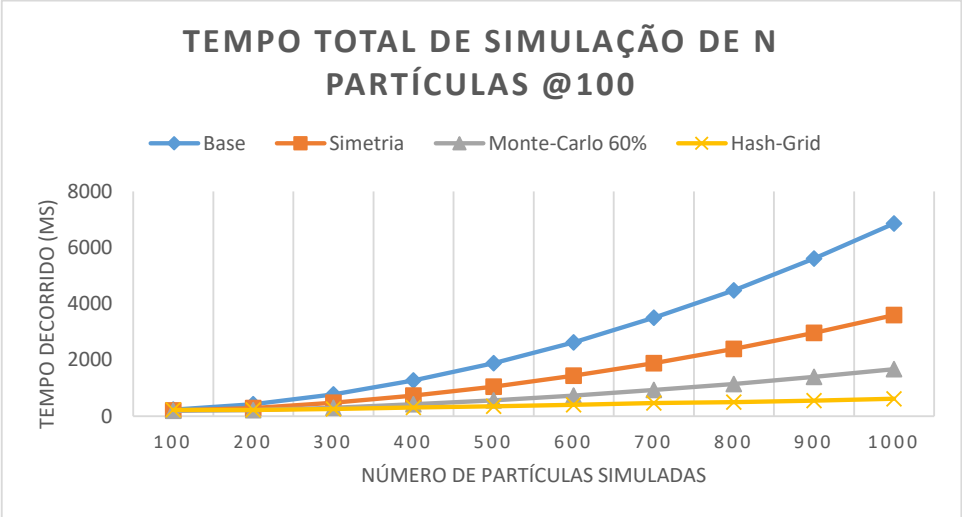


Figura 5.2: Tempo total, decorrido durante todas as etapas de simulação do fluido, nos primeiros 100 quadros e para cada quantidade de partículas a simular.

momento em que se inicia a instanciação das partículas, até ao momento em que a simulação atinge os 100 *frames*[1], para cada quantidade de partículas. Este gráfico reforça, mais uma vez, o que se tinha concluído anteriormente. Apesar de inicialmente, o Algo.4 (identificado no gráfico como Hash-Grid) ser mais lento que a maioria dos restantes, com o aumento da quantidade de partículas a simular, esses mesmos algoritmos, sofrem um aumento do tempo de simulação bastante mais acentuado que o algoritmo enunciado.

Dos dois gráficos apresentados nas Figuras 5.1 e 5.2, ainda é possível observar que,

[1]O motivo que levou à decisão de limitar esta simulação nos 100 *frames*, reside no facto de que, para calcular o momento em que a velocidade média das partículas se aproxima de 0, resultaria na redução de desempenho da simulação, deste modo para que a simulação de menos partículas não ficasse estática por muito tempo, o valor ótimo, para o limite de *frames*, comprovou-se sendo os 100 selecionados.

como era de se esperar, com o aumento do número de partículas a simular, o Algo.1 (identificado no gráfico como algoritmo Base), por computar a interação das partículas todas com todas, em ambos os sentidos, seria aquele que ia apresentar um aumento mais acentuado do grau de inclinação da função exponencial que o define. Por outro lado, o crescimento da função do Algo.4 aparenta ser praticamente linear, não sofrendo grandes desvios no seu fator de crescimento.

Passando para os restantes dois algoritmos, estes aparentam ser os mais idênticos entre os quatro. Era de se esperar que o Algo.3 (identificado no gráfico como algoritmo Monte-Carlo 60%) apresentasse uma otimização relativamente melhor que a aquela que se observa entre os dois, já que o de Monte-Carlo, para além de conter a otimização implementada pelo Algo.2 (identificado no gráfico como algoritmo Simetria), ainda tem a particularidade de apenas uma certa percentagem de interações ser considerada. O que se sucede, é que para o gráfico da Figura apresentada, foi escolhida uma percentagem de interações de 60%, por ser a percentagem que apresentava uma melhor relação de eficiência/precisão. Posteriormente, na secção 5.2.4, estas propriedades do algoritmo de Monte-Carlo, serão exploradas mais afundo.

5.2.2 Otimizações de Simetria de Forças

Com a perspectiva de melhorar os resultados obtidos das simulações com o Algo.2 (Simetria) e Algo.3 (Monte-Carlo), foi considerada uma pequena otimização nos cálculos das forças causadas pela interação entre partículas. A otimização considerada, foi inspirada por Muller et al. [34]. Esta otimização, tal como referido na secção 4.1.2, consiste em reduzir a quantidade de cálculos efetuados, considerando que a força atuada numa partícula é igual à partícula com a qual está a interagir. Ao simular estas otimizações, chegamos aos gráficos das Figuras 5.3 e 5.4. Apesar de promissor, ao analisar estes gráficos é possível concluir que estes cálculos já são tão eficientes, que esta simples otimização se torna redundante em comparação à redução de iterações dos algoritmos propostos na tabela 5.3.

5.2.3 Distribuição dos Intervalos de Tempo

Após a análise dos resultados demonstrados no decorrer das secções anteriores, é importante entender como é que estes resultados são divididos pelas diferentes fases do simulador. Para esta divisão foram destacadas duas etapas, a etapa de instanciação das partículas, e a segunda etapa, onde todas as restantes computações foram consideradas como parte da simulação do comportamento. A única exceção a esta divisão, foi feita para o Algo.4, em que para a primeira etapa, para além do tempo de instanciação, ainda é acrescentado o tempo de criação da grelha de simulação. Nas Figuras 5.5 e 5.6 é possível verificar os resultados obtidos da divisão proposta.

Interpretando os resultados observados na Figura 5.6, conclui-se que embora a relação entre os tempos de simulação dos diferentes algoritmos não divergir demasiado em

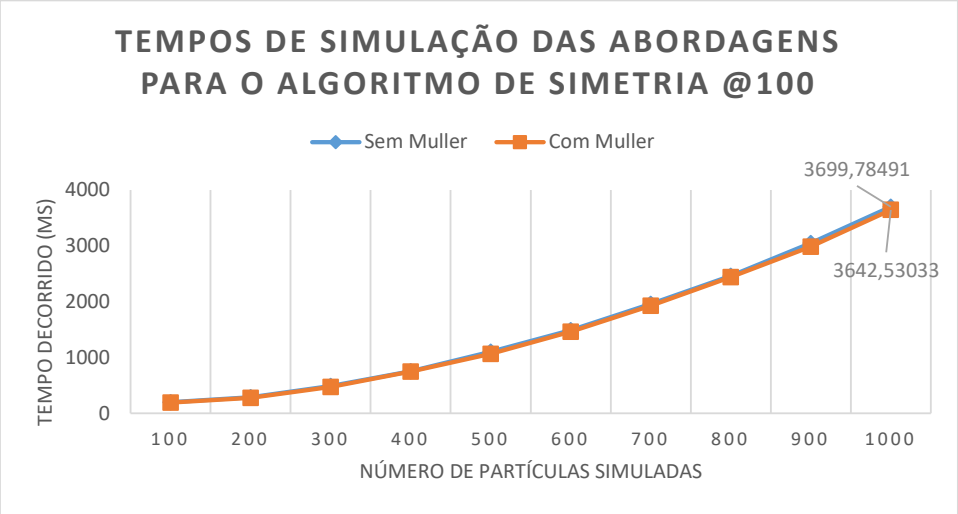


Figura 5.3: Tempo decorrido ao simular as partículas com o algoritmo de "Simetria"nos 100 primeiros *frames*, comparando os resultados das duas abordagens, com e sem simetria das forças. Nota:Valores acima das retas, representam a abordagem sem Otimização de Muller. Valores abaixo das retas, representam a abordagem com Otimização de Muller.

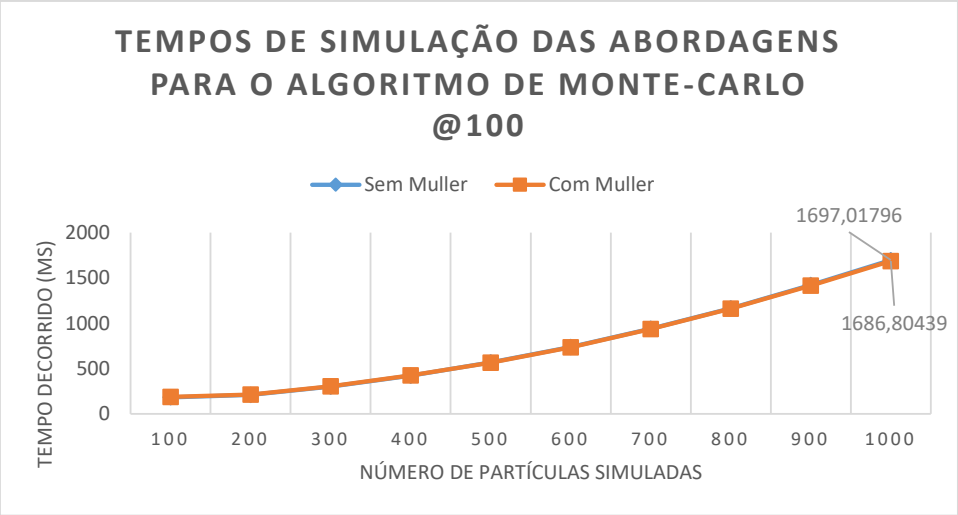


Figura 5.4: Tempo decorrido ao simular as partículas com o algoritmo de "Monte-Carlo"nos 100 primeiros *frames*, comparando os resultados das duas abordagens, com e sem simetria das forças. Nota:Valores acima das retas, representam a abordagem sem Otimização de Muller. Valores abaixo das retas, representam a abordagem com Otimização de Muller.

comparação com o que se observou na secção anterior, o mesmo deixa de ser observado na análise do tempo de instanciação das partículas da Figura 5.5. O que se verifica é que apesar dos resultados dos algoritmos, Base, Simetria e Monte-Carlo 60%, apresentarem intervalos de tempo idênticos, o mesmo deixa de se comprovar nos tempos de instanciação para o algoritmo otimizado com uma *Hash-Grid*. O que se conclui, é que este fator pode ser um dos motivos a contribuir para que a simulação com o nosso Algo.4, perante uma quantidade reduzida de partículas, não se destacar em relação às ademais simulações.

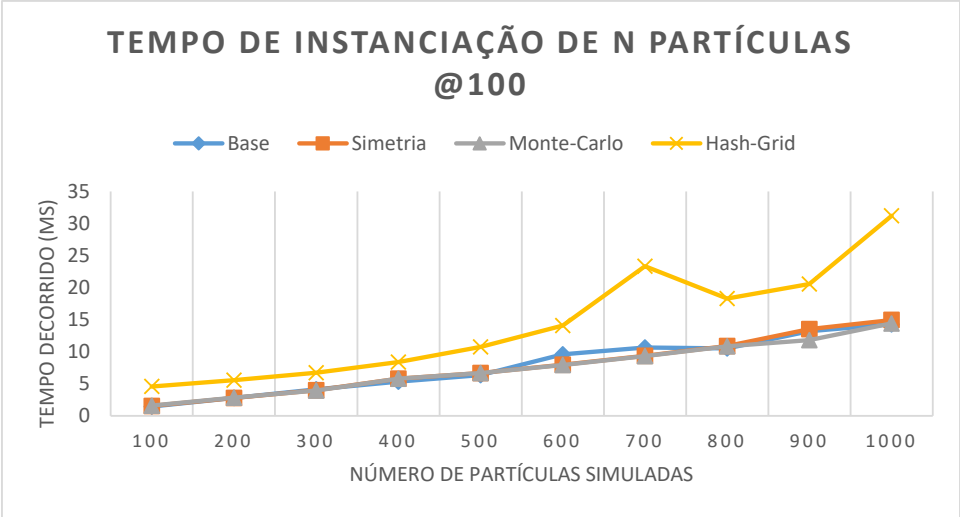


Figura 5.5: Tempo decorrido na etapa de instanciação das partículas, considerando diferentes quantidades de partículas a simular nos primeiros 100 quadros da simulação.

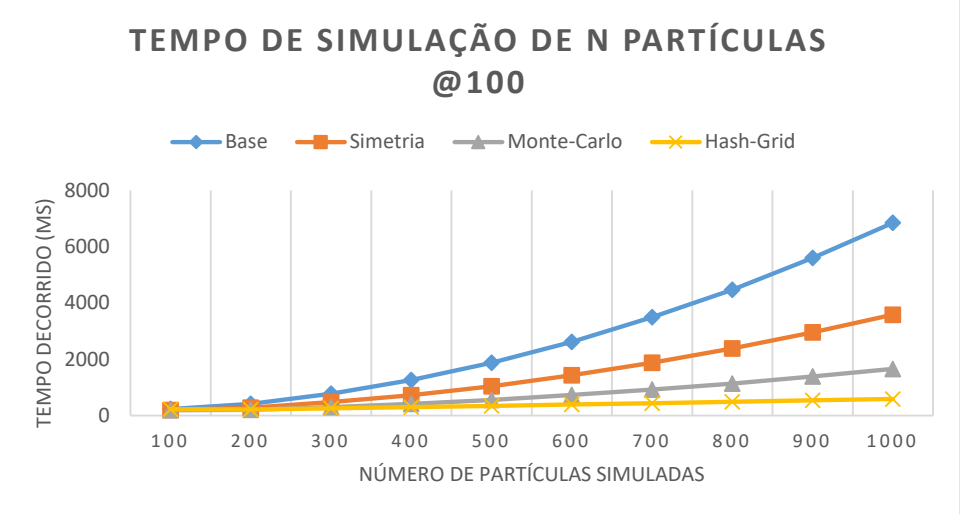


Figura 5.6: Tempo decorrido na etapa de simulação das partículas, considerando diferentes quantidades de partículas a simular nos primeiros 100 quadros da simulação.

5.2.4 Otimização de Monte-Carlo

Anteriormente, foi possível realizar a apreciação geral dos resultados obtidos entre os diferentes algoritmos implementados. Nesta secção, serão analisadas algumas das propriedades mais especificas que a otimização de Monte-Carlo apresenta.

Desses resultados, como já fora apresentado nas secções anteriores, apesar de Monte-Carlo ser mais eficiente que a otimização por simetria, este aparenta ficar um pouco abaixo das expetativas. Como forma de esclarecer o que origina estes resultados, procedeu-se à simulação, de exatamente o mesmo fluido, mas desta vez, apenas com a otimização de Monte-Carlo e recorrendo a diferentes percentagens de interações consideradas.

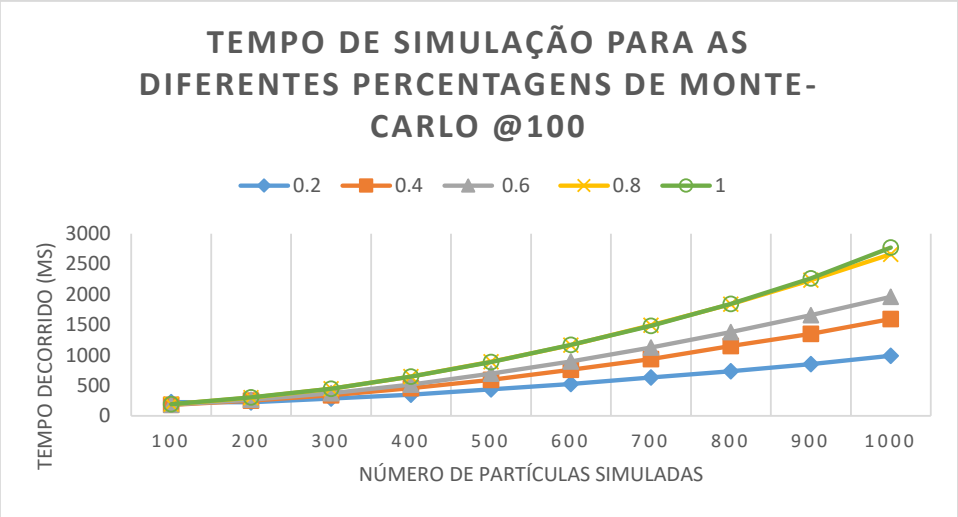


Figura 5.7: Tempo decorrido a executar os primeiros 100 quandros da simulação de Monte-Carlo dependendo da percentagem de interações a considerar.

As percentagens selecionadas foram entre o intervalo de 20% a 100%, com saltos de 20% entre cada uma das simulações. Os resultados para estas demonstrações são exibidos no gráfico da Figura 5.7. Apesar de, ao analisar o gráfico, apenas se qualificar a otimização quanto à sua velocidade de computação, existe outra condicionante que exerce um enorme peso na escolha da melhor percentagem de interações. Esta referida condicionante, corresponde à precisão dos resultados, que impacta diretamente o comportamento que se observa do fluido. Assim, se apenas considerarmos o gráfico da Figura 5.7 e a tabela 5.4, rapidamente se afirmava que a melhor escolha seria apenas considerar os 20% das interações. Todavia, ao considerar o que se observa na Figura 5.8, instantaneamente se muda de opinião em relação a esta percentagem.

Na Figura 5.8 observa-se um fenómeno que, embora crítico para uma correta simulação do fluido que se pretende, permite que se retire informações bastante interessantes. O que está a provocar o comportamento que se observa nesta imagem, é que, a baixa percentagem de interações a considerar, causa um enorme défice das forças a atuar sobre cada partícula, provocando os erros de conservação da massa que se observa. Com base nos resultados apresentados, verificou-se que o melhor valor para a percentagem de interações seria o de 60%, excluída a hipótese em que a simulação pretendida pelo utilizador não beneficie da precisão dos resultados gerados.

5.2.5 Otimização Mista

No decorrer deste capítulo tem sido reforçada a ideia que, de modo geral, quando a simulação é composta por uma elevada quantidade de partículas, a otimização com recurso a Hash-Grid (Algo.4) será a mais indicada. Como tal, nesta secção, será feita uma análise mais detalhada, sobre esta otimização e das suas componentes.

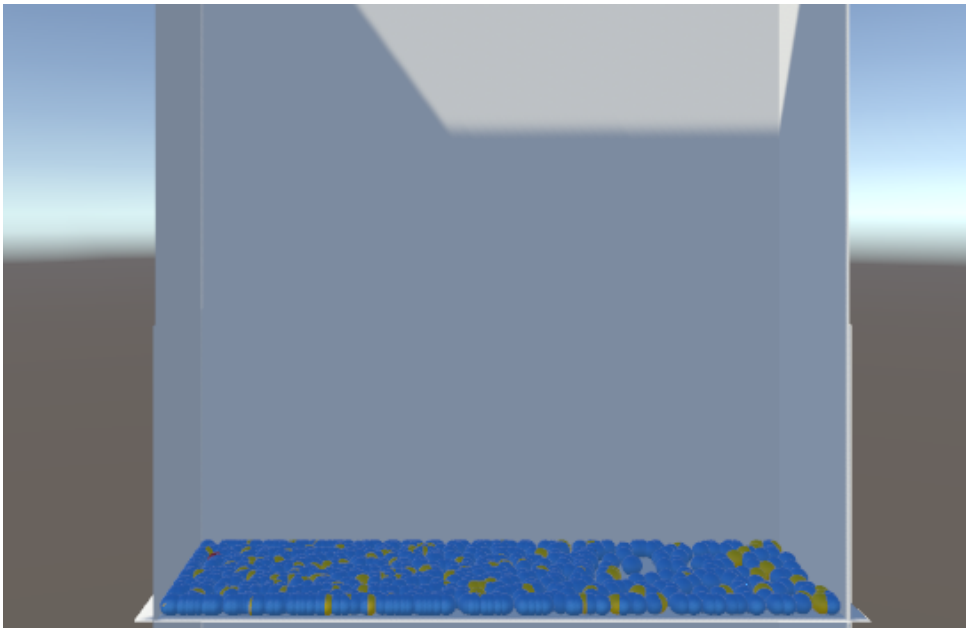


Figura 5.8: Aglomerado de partículas causado pelo défice de interações, considerando apenas 20% das interações de Monte-Carlo.

Após a análise do gráfico da Figura 5.2, concluiu-se que a simulação de uma quantidade reduzida de partículas seria o ponto fraco desta otimização. De seguida, com recurso ao gráfico da Figura 5.5, foi possível observar, que um potencial fator que levasse a este resultado não estaria presente na simulação do fluido, mas sim, no tempo de preparação da cena e instanciação das partículas do fluido. Assim, de modo a compreender de que forma estas etapas podem afetar os resultados da nossa simulação, começou-se por investigar melhor a computação destas etapas. Anteriormente, na secção 4.1.4, é explicado que, para este tipo de otimização, é necessário que se recorra à criação de uma grelha auxiliar, onde, por sua vez, serão mapeadas as posições de cada partícula. Como tal, na nossa implementação, é possível criar a dita grelha com as dimensões que o utilizador considerar mais adequadas, tendo de manter em consideração que, quanto maior for a grelha, pior será o rendimento da simulação. Estas dimensões dependem dos valores das dimensões escolhidas para os eixos **x**, **y**, **z**, e a razão de impacto que estas dimensões causam nos resultados da simulação. Este fator de dimensão pode ser observado no gráfico da Figura 5.9.

Começando por analisar o gráfico da Figura 5.9, verifica-se que o tempo de processamento de um determinado número de partículas aumenta, na medida em que o tamanho da grelha de simulação também aumenta. Seria espectável que esta situação acontecesse, dado que com o aumento das dimensões da grelha, seria gerada uma maior quantidade de células a processar e consequentemente a alocação em memória, provocando um aumento no tempo de simulação de dito fluido. Apesar da variação ser reduzida, a tendência é que com o aumento do número de partículas, o tempo de processamento aumente a um ritmo mais elevado para as grelhas de maior dimensão. Isto pode derivar do aumento da

quantidade de células a processar e do processo da sua alocação aumentar de nível de complexidade.

Na Figura 5.10, é possível concluir que, o tempo de processamento da grelha seja um fator significante no tempo médio de processamento das diferentes dimensões de grelha. Conforme o aumento do número de partículas e dimensões da grelha, o seu tempo de processamento torna-se um fator cada vez mais significativo. O que se conclui que, o tempo de computação depende diretamente das suas dimensões, tornando a escolha apropriada das dimensões da grelha, um fator com extremo impacto nos resultados da simulação.

Relativamente ao gráfico da Figura 5.11, é possível observar que, o tempo de alocação das partículas é relativamente consistente na simulação das diferentes quantidades de partículas. Contudo, com o aumento das dimensões da grelha, o tempo de alocação aumenta levemente. Desta análise, é seguro concluir que o aumento da grelha e o número de partículas não são suficientemente relevantes para os resultados da simulação.

Para concluir a análise desta otimização, é necessário que se considere uma última propriedade que a nossa implementação coloca na computação do fluido. Deste modo, para além dos fatores apresentados, esta otimização é condicionada a uma área de simulação estática, ou seja, caso o fluido a simular se desloque, excedendo os limites da grelha que lhe foi imposta, resulta na saída da área de simulação e provoca uma falha crítica no nosso simulador.

5.3 Diferentes Comportamentos do Fluido

Na criação de uma simulação de fluidos, como estudado no capítulo 2, existe uma vasta diversidade de fluidos, comportamentos e, inclusivamente, cenários de simulação. Como tal, também as abordagens são o mais diversificadas que se possam imaginar. Nesta secção, serão apresentados os diversos comportamentos, fluidos, cenários e modos de observação, incluídos pela nossa implementação.

5.3.1 Observação de Cenários

Na secção 4.2.1 foram apresentados os quatro cenários em que se focou a implementação do nosso simulador de fluidos. De seguida, será finalmente possível verificar como é que os nossos fluidos se comportam quando colocados à prova nos diferentes cenários disponíveis. Na tabela 5.5 encontra-se a lista dos diferentes cenários disponíveis.

Começando pelo cenário, C1, o propósito consistia em testar a queda de um volume de fluido. Com este simples teste, foi possível, não só testar o efeito que a força da gravidade causa no volume como um todo, mas também, a reação das partículas ao entrar em contacto com uma superfície, e de que forma estas mesmas partículas se movimentam nesse plano, até que a velocidade do fluido chegue a zero. Na sequência de Figuras que se seguem na Figura 5.12, é possível observar as várias etapas acima referidas.

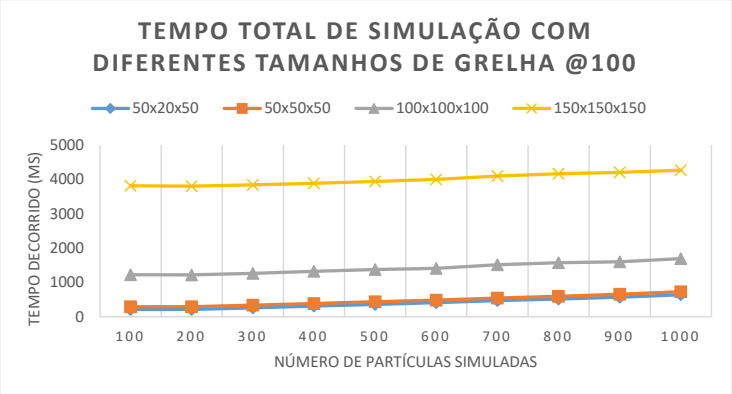


Figura 5.9: Tempo decorrido durante nos primeiros 100 quadros da simulação do fluido, considerando diferentes dimensões de grelha e número de partículas.

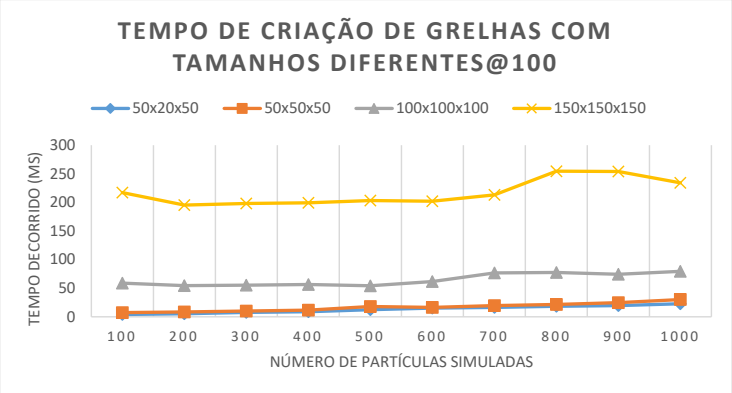


Figura 5.10: Tempo decorrido na etapa de contrução da grelha, considerando diferentes dimensões de grelha e número de partículas.

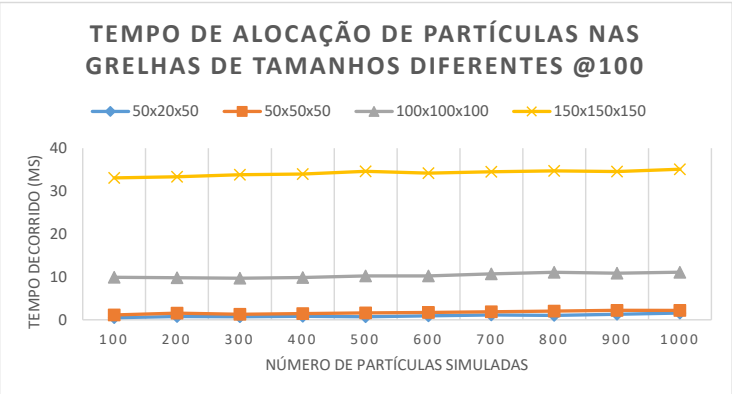
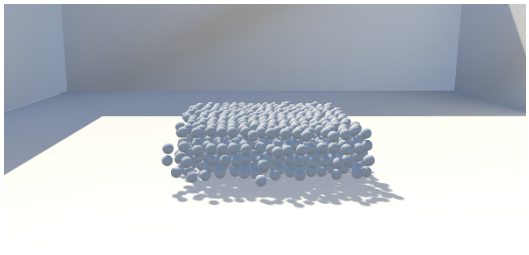


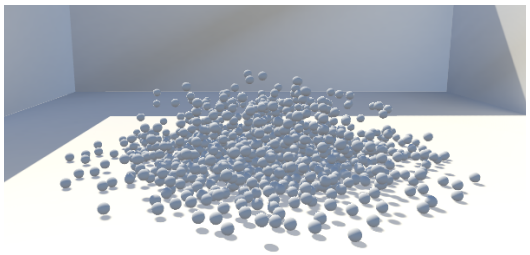
Figura 5.11: Tempo decorrido na etapa de alocação das partículas na grelha, considerando diferentes dimensões da grelha e número de partículas.

Tabela 5.5: Cenários de teste ao comportamento de fluidos.

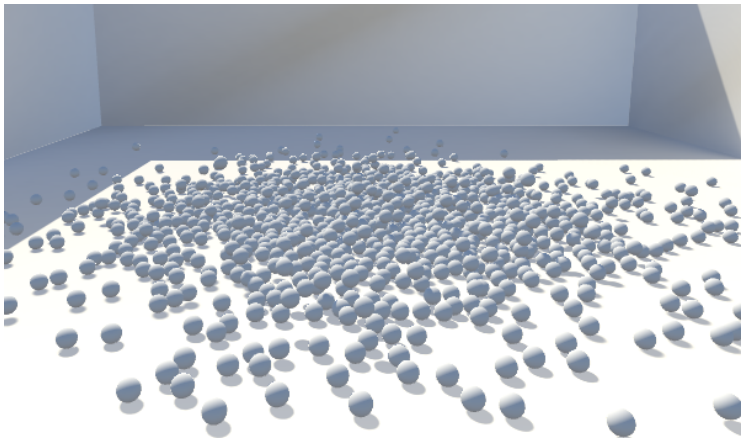
	Cenários
C1	Caixa simples
C2	Plano inclinado
C3	Câmara de ondas
C4	Câmara com obstáculos



(a) Iniciação das partículas.



(b) Contacto com superficie.

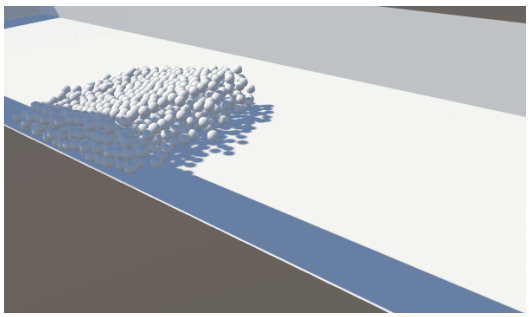


(c) Dispersão final de partículas.

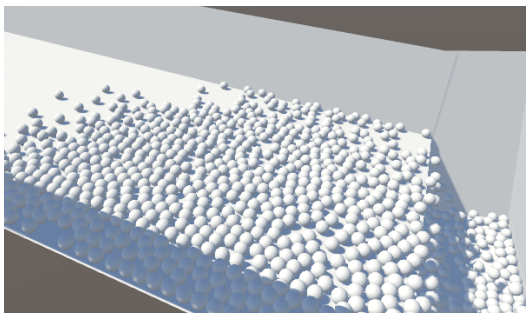
Figura 5.12: Etapas de interação do fluido na Caixa simples.

Prosseguindo com os cenários apresentados, segue-se, C2, um plano inclinado com um bloco vazio no final. O que se pretendeu com este cenário, foi observar de que forma as partículas se movimentam ao longo de toda a superfície, e principalmente, observar o efeito de cascata causado pelo final do plano, incluindo o efeito originado pela queda do fluido sobre o restante fluido já presente no bloco final do cenário. De seguida, na Figura 5.13, é possível observar, estes mesmos efeitos. Como referido, um dos efeitos que se pretende testemunhar, é o comportamento do fluido em cascata ao entrar em contacto com o restante fluido. Deste modo, para auxiliar a observação destes mesmos comportamentos, na Figura 5.13d, é possível verificar o impacto provocado pela interação destes fluidos e assim, analisar a coerência dos resultados obtidos.

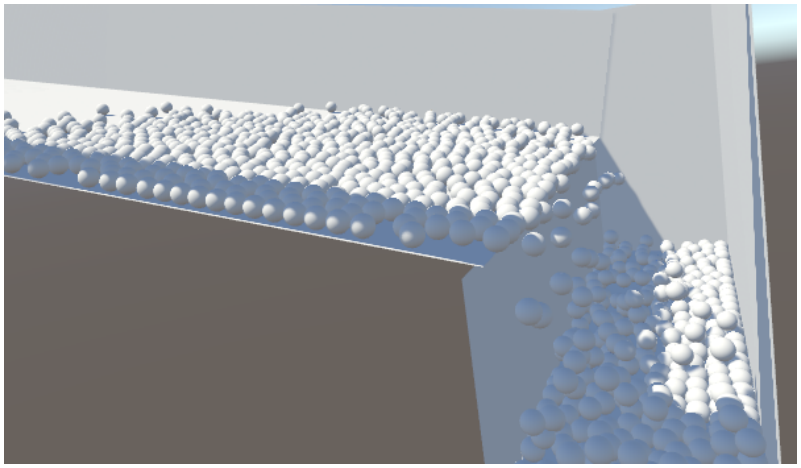
Interpretando o que se observa na Figura 5.13d, compreende-se que a queda das partículas no restante fluido, causa o aumento da sua velocidade, que origina um efeito de espuma e deslocação do fluido, no volume de contacto com o fluido em queda.



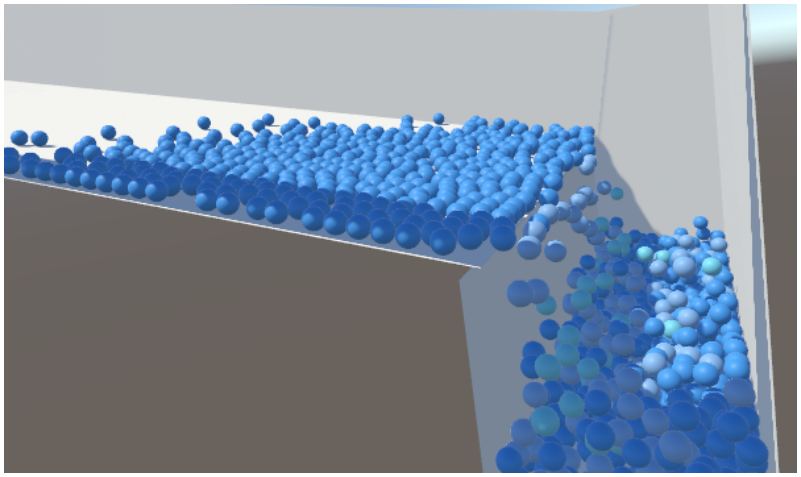
(a) Iniciação das partículas.



(b) Contacto com superficie e fluidez.



(c) Efeito em cascata.



(d) Figura onde é possível observar o efeito em cascata promovido pelo simulador de fluidos implementado.

Figura 5.13: Etapas de interação do fluido no Plano inclinado.

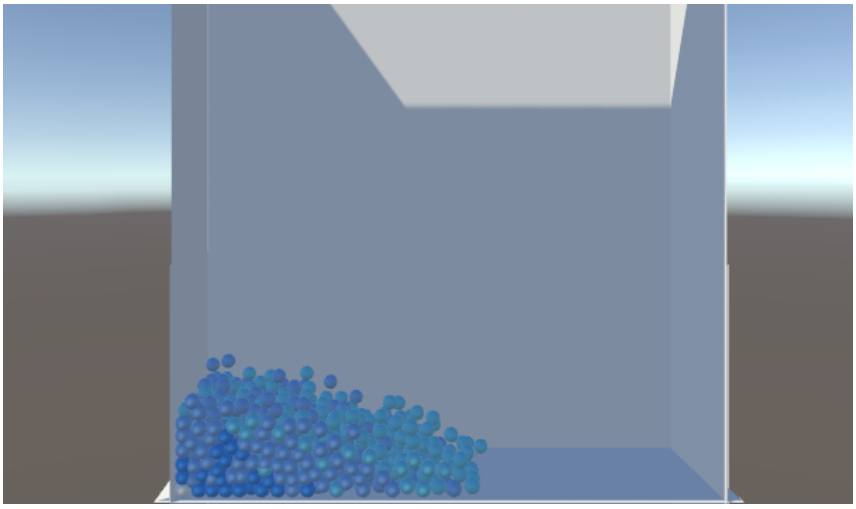
Prosseguindo a análise dos cenários implementados, nas Figuras 5.14 e 5.15, mostramos a interação do fluido com o cenário C3. Neste cenário é provocada a interação do nosso fluido com as placas móveis e deste modo, é forçada a interação entre partículas do mesmo cenário e aumentada a pressão aplicada sobre o fluido. Na base da criação deste cenário, esteve a intenção de testar a formação de ondas no volume total do fluido simulado. De modo a facilitar a recolha de informação, tal como se apresentou no cenário anterior, faz-se mais uma vez, recurso ao efeito de espuma apresentado na secção 4.2.2.

Nas Figuras 5.14 e 5.15 , estão divididas as diferentes etapas do comportamento do fluido colocado em simulação no cenário C3. Esta divisão começa por 5.14a, onde é possível observar o comportamento do fluido ao entrar na câmara de ondas e, consequentemente, a formação do tubo de fluido formado no momento de impacto com o limite lateral da cena 5.14b. De seguida, sem a interferência de forças exteriores, o nosso fluido começa a estabilizar até que a sua superfície atinja um estado completamente uniforme, resultando no volume que se observa na Figura 5.14c. Neste momento, é ativada a movimentação da primeira placa lateral, provocando a movimentação e interação entre partículas. Deste modo, é originada a pequena ondulação que pode ser observada nas Figuras 5.15a e 5.15b. Quando se coloca uma placa a pressionar um liquido, é de se esperar que a movimentação causada no mesmo dependa da força e velocidade que a placa lhe coloca. Como tal, foi criada uma segunda placa, mas com a força e velocidade muito mais expressivas, para confirmar o impacto que diferentes forças externas aplicam no comportamento do fluido. Da interação com esta placa resulta a movimentação observada nas Figuras 5.15c e 5.15d.

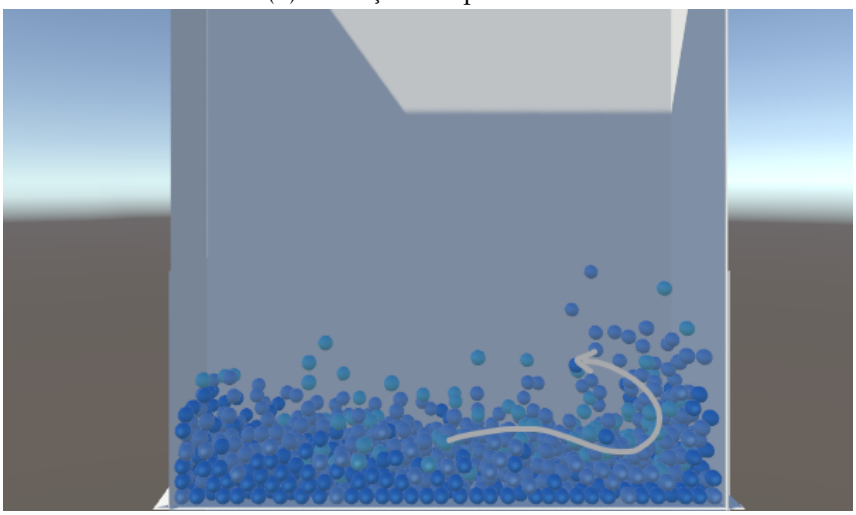
Para finalizar a análise dos cenários implementados, segue-se o cenário C4, construído com o objetivo de reforçar o estudo do comportamento do fluido ao interagir com diversos obstáculos. Assim, partiu-se da construção base do cenário C3, com a adição de uma subdivisão do espaço. Com este cenário, perspetivou-se observar o impacto que um obstáculo colocado na movimentação previsível do fluido pode causar ao comportamento normal do mesmo. Deste modo, ao observar a Figura 5.16, é possível concluir que, não só o fluido interage com o obstáculo provocando um grande aumento inicial de pressão nas partículas, como por resultado dessa mesma pressão, o fluido contorna o obstáculo como previsto e preenche o volume do cenário até que as zonas de pressão sejam estabilizadas, resultando num fluido completamente estático. Para além disso, é importante alertar que a simulação do ar não foi implementada, deste modo, não temos um sistema fechado de vasos comunicantes, o que pelos princípios de Stevin seria garantido que a altura do fluido, em ambos os lados do obstáculo, seria sempre a mesma, uma vez que a pressão na superfície do fluido seria a pressão atmosférica.

5.3.2 Fluidos e suas Propriedades

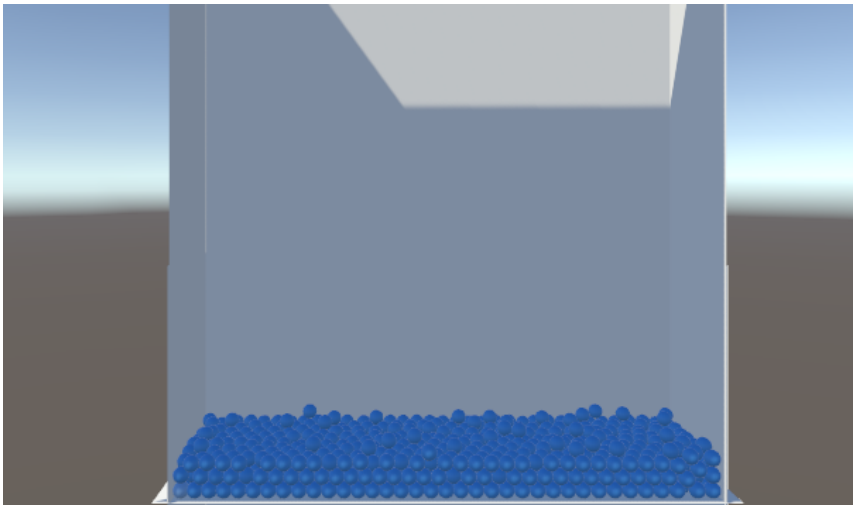
Como tem vindo a ser reforçado até ao momento, existem diversos fluidos diferentes, implicando assim, a necessidade de simular cada um desses fluidos e respetivas propriedades. Ao relembrar a tabela 4.1, é possível verificar a lista de propriedades que o nosso



(a) Iniciação das partículas.

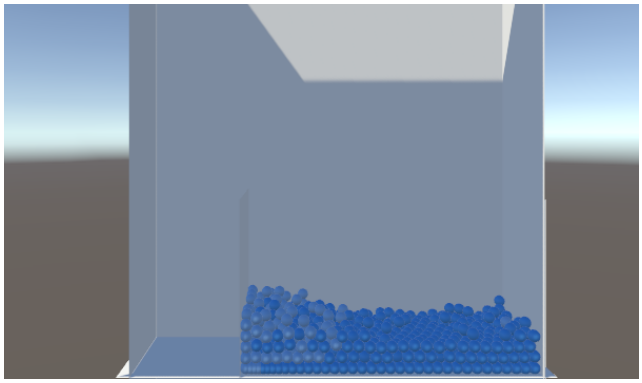


(b) Efeito de criação de tubo.

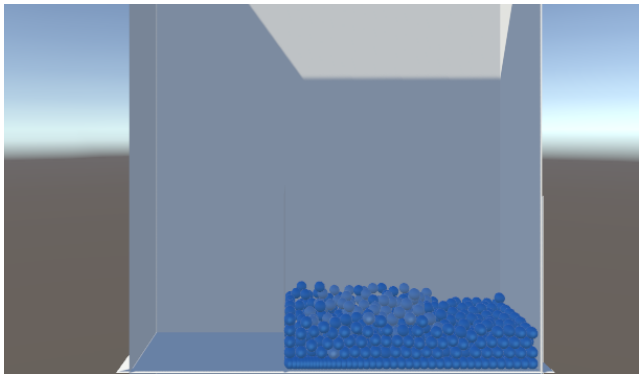


(c) Partículas estabilizadas.

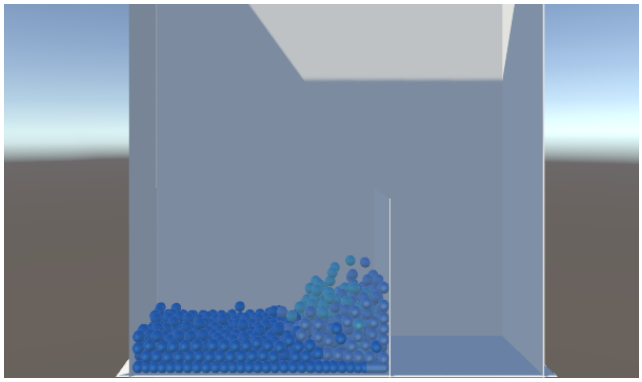
Figura 5.14: Etapas de interação do fluido na Câmara de ondas.



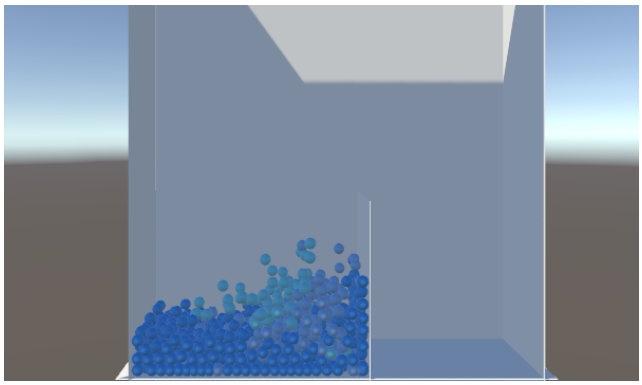
(a) Efeito causado pela interação com a primeira placa.



(b) Ondulação causada pela primeira placa.

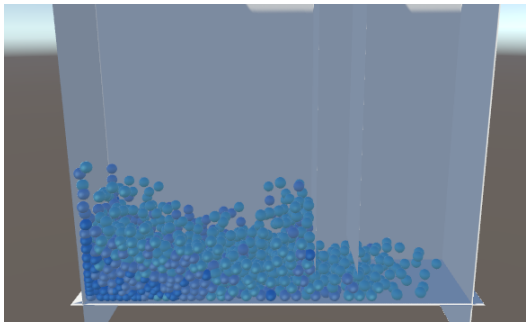


(c) Efeito causado pela interação com a segunda placa.

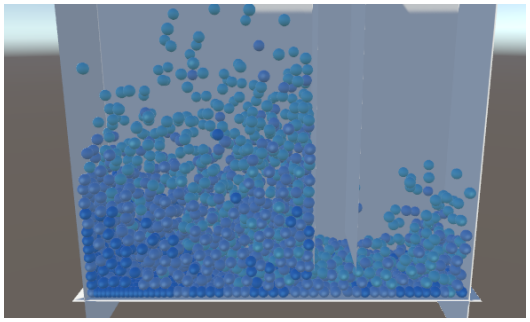


(d) Ondulação causada pela segunda placa.

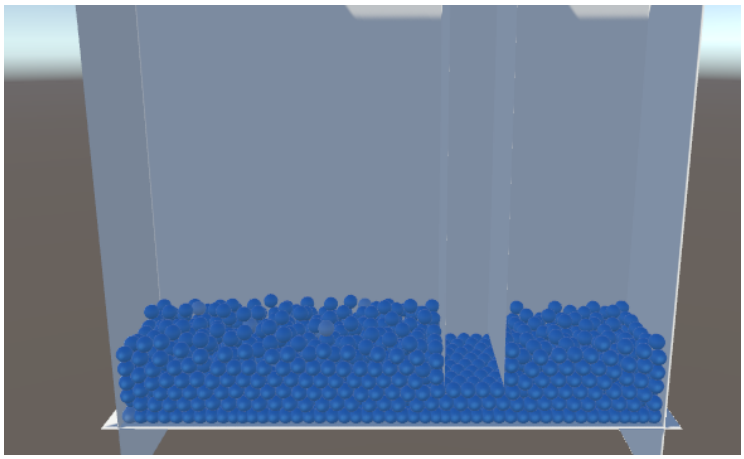
Figura 5.15: Etapas de interação do fluido na Câmara de ondas, ao interagir com a placa em movimento.



(a) Iniciação das partículas e movimentação inicial.



(b) Contacto repentino com um obstáculo e consequente aumento de pressão.



(c) Estabilização do fluido ao contornar o obstáculo.

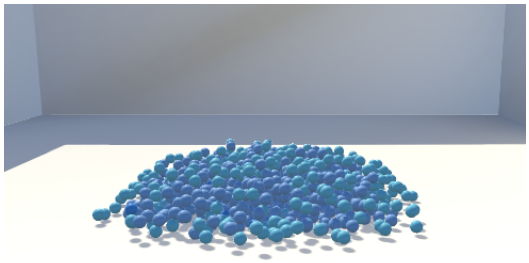
Figura 5.16: Etapas de interação do fluido na Câmara com obstáculos.

simulador disponibiliza, e desta forma, possibilitar a simulação de diferentes fluidos líquidos. Nesta secção serão apresentados diferentes fluidos e o resultado da sua interação e comportamento. Apesar de diversas propriedades, nem todas possibilitam uma análise simples das duas diferenças, como tal, o foco será maioritariamente em fluidos que apresentem comportamentos com diferentes graus de viscosidade.

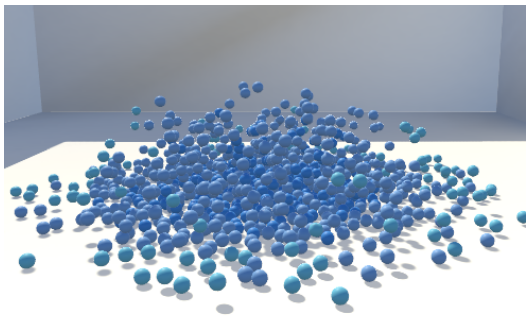
Fluidos com diferentes níveis de viscosidade e consequente amortecimento dos movimentos, sempre será uma propriedade crucial para simulações de fluidos, em aplicação como os vídeo jogos. A abordagem que prevalece para a simulação destes fluidos viscosos é baseada nas equações de Navier-Stokes, e deste modo, também a nossa implementação é capaz de os simular. Para meter esta propriedade à prova, foram selecionados vários conjuntos de propriedades, resultando em fluidos com comportamentos que comprovam o funcionamento das diferentes viscosidades que um fluido pode apresentar. Assim, nas Figuras 5.17 e 5.18 é possível verificar os respetivos comportamentos.

5.4 Discussão

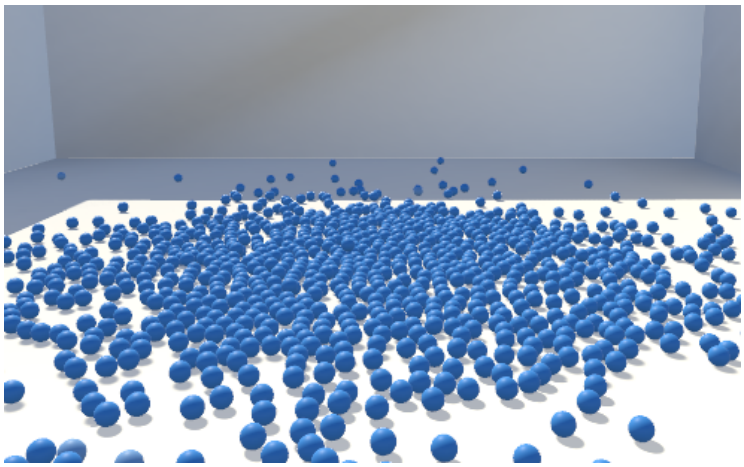
Em suma, as avaliações realizadas ao nosso Sistema de Simulação de Fluidos revelaram resultados positivos e bastante interessantes para as otimizações implementadas.



(a) Iniciação e queda das partículas.



(b) Contacto com superficie.

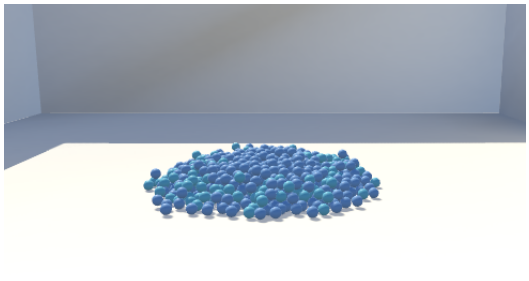


(c) Dispersão final das partículas.

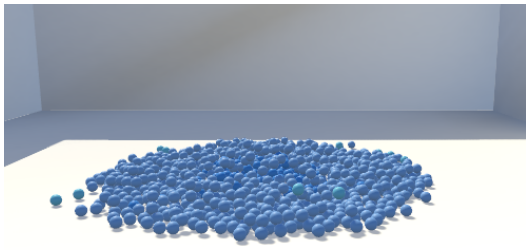
Figura 5.17: Comportamento e interação das partículas de um fluido com *Rest density* de 1000 e viscosidade de 1 procurando simular um fluido menos viscoso, como é o caso da água.

Com a implementação destas otimizações, pretendemos que para além de melhorarem o desempenho da simulação, também possibilitem a integração de Simulações de Fluidos em diferentes cenários de simulação.

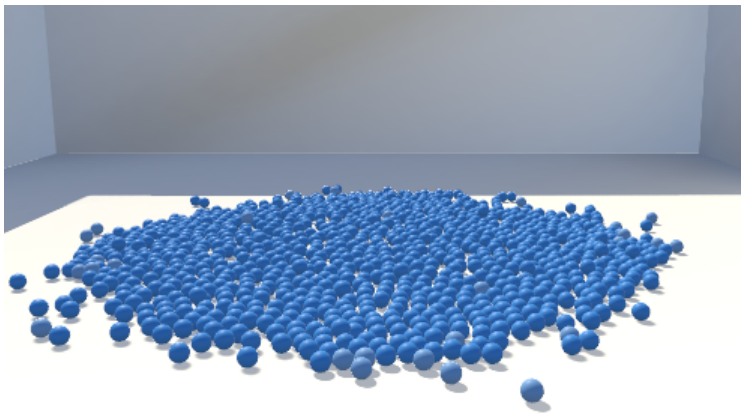
Na avaliação dos resultados obtidos da secção 5.2, são retiradas diversas conclusões em relação aos algoritmos de otimização implementados. Em relação ao algoritmo utilizado como base, facilmente se conclui que, tirando um cenário muito hipotético em que o número de partículas e/ou a quantidade de FPS sejam muito reduzidos, dificilmente este algoritmo será a melhor opção. De seguida, foram testados os algoritmos de otimização que focam apenas na interação pura entre partículas. O primeiro, trata-se do algoritmo de otimização Simetria de forças, que foi comprovado na secção 5.2.2, que o real impacto no melhoramento de rendimento da simulação está na redução de interações a computar, e que otimizar o modo que as forças são calculadas, como descrito na secção 5.2, não tem um impacto realmente significativo nos resultados obtidos. Apesar desta otimização já obter resultados mais satisfatórios, esta continua a ser mais indicada para simulações com uma quantidade de partículas relativamente reduzida. Deste modo, com a otimização



(a) Iniciação e queda das partículas.



(b) Contacto com superficie.



(c) Dispersão final das partículas.

Figura 5.18: Comportamento e interação das partículas de um fluido com *Rest density* de 1500 e viscosidade de 1000 procurando simular um fluido mais viscoso, como o mel ou a lava.

do algoritmo anterior, por via de Simulações de Monte-Carlo, na secção 5.2.4, são apresentados resultados que mostram que dependendo do grau de precisão que o utilizador está disposto a sacrificar, é possível obter tempos de computação bastante aceitáveis. Por fim, com base nos resultados obtidos na secção 5.2.5, conclui-se que, em termos de rendimento, quando é necessária uma quantidade superior de partículas, a melhor otimização é a otimização com adaptação híbrida, dependendo apenas do tamanho da grelha de simulação. Em suma, a otimização mais rentável é a nossa otimização com adaptação híbrida. Todavia, se o volume do fluido se deslocar numa cena de grandes dimensões, torna-se inconcebível a criação de uma grelha demasiado grande. Como a nossa implementação é constituída por uma grelha estática, conclui-se que, para um cenário como este, a melhor otimização estará entre a otimização por Simetria de Forças e a otimização por Monte-Carlo.

Relativamente à avaliação dos resultados obtidos da observação dos cenários, na secção 5.3, é descrito o comportamento suportado pelo nosso fluido quando simulado nos diferentes cenários. Na Simulação de Fluidos é importante que os fluidos simulados sejam capazes de comportar diversos tipos de simulação. Assim, como é descrito na secção 5.3.1, estão implementados quatro cenários que colocam à prova diferentes propriedades de

um fluido. No primeiro cenário, Caixa Simples, é possível testar a queda de um volume de fluido e observar de que forma é que este reage ao colidir com o cenário. O comportamento que se observa é semelhante a um grande salpico de fluido que pode comportar diversas propriedades. Na secção 5.3.2, são testadas diferentes propriedades do fluido a simular, deste modo, são apresentados dois fluidos que apresentam diferentes viscosidades. Com os fluidos apresentados, a intenção é simular a água e um fluido mais viscoso, como o mel, e assim é verificado que o nosso simulador consegue simular fluidos com propriedades completamente distintas. De seguida, com o cenário Plano Inclinado, é possível observar diversos comportamentos. Entre estes comportamentos, verifica-se mais uma vez, o fluxo dos diferentes fluidos ao percorrer a superfície do plano. Para além disso, observa-se um comportamento muito interessante quando o fluido chega ao final da rampa, onde é simulado um efeito em cascata, tal como previsto por um comportamento normal de um fluido real. No terceiro cenário, identificado por Câmara de Ondas, é simulada a interação do fluido com outros planos em movimento. De notar, que esta interação é computada em tempo real, e como tal, para que o fluido interaja com os planos em movimento, ambos os objetos têm de estar dependentes do mesmo passo de tempo, para que as partículas do fluido consigam computar a interação com o dito plano. Além disso, é interessante constatar que neste cenário, o fluido se comporta como a ondulação do mar. Contudo, se o fluido for computado a uma frequência muito reduzida de quadros por segundo FPS, a interação com o plano pode causar problemas, uma vez que alguns dos contactos entre partículas e o plano podem acabar por ser ignorados. Por fim, no cenário Câmara com obstáculos, a colisão com diferentes obstáculos é posta novamente à prova. Apesar da simulação do ar não ter sido implementada, e deste modo, impossibilitando a construção de um sistema fechado de vasos comunicantes, este cenário comprova que os fluidos simulados são capazes de interagir com diferentes obstáculos em simultâneo, adaptando-se às potenciais restrições e eventos que possam ocorrer no ambiente de simulação em que estão inseridos.

6

Conclusão

Simulação de fluidos, é composta por um conjunto de técnicas e ferramentas usadas no estudo, implementação e manipulação de fluidos virtuais num ambiente gerado por computadores. Recentemente, estas simulações, têm-se tornado cada vez mais relevantes, com a capacidade de resolver problemas em áreas como a engenharia aeroespacial, automobilismo, energia e biomedicina. Para além disso, fluidos como a água, o ar, o fogo, e outros gases, podem desempenhar um papel crucial em múltiplos efeitos especiais incluídos em filmes e vídeo jogos.

Entre os diversos sectores anunciados, o mercado do entretenimento, com os vídeo jogos, contribuiu para um enorme crescimento da demanda por efeitos físicos e visuais cada vez mais realistas e dinâmicos. Isto levou a diversos avanços na investigação de Simulações de Fluidos com o objetivo de alcançar técnicas de simulação mais avançadas. Uma vez que os seus gráficos se têm tornado mais sofisticados, desenvolvedores de jogos continuam a superar os limites destas simulações com técnicas de simulação em tempo real, largamente utilizadas em alguns dos jogos mais conhecidos, como é o caso do "Half-Life 2"(2004) e o "Uncharted 2: Among Thieves"(2009). Como resultado desta requisição, uma das suas maiores contribuições foi a criação do método de simulação *Smoothed Particles Hydrodynamics*(SPH), um dos métodos de Simulação de Fluidos mais utilizado e que foi o método de simulação implementado na nossa investigação.

Com os grandes avanços que motores de jogo como o Unity têm vindo a receber, esta dissertação foca-se na implementação de um sistema de Simulação de Fluidos, com múltiplas otimizações que possibilitem a criação de simulações físicas de um fluido em aplicações como vídeo jogos. Deste modo, a presente dissertação discute esta possível necessidade, abordando as questões de investigação estabelecidas na secção 1.2. Assim, com os objetivos delineados na secção 1.3, implementamos no Unity3D, um método de simulação de fluidos baseado em *Smoothed Particles Hydrodynamics*(SPH) acompanhado de 3 otimizações diferentes para a procura das partículas em interação. Por sua vez, estes algoritmos e implementação estão explicados em maior detalhe nos capítulos 3 e 4.

Após considerar os resultados das avaliações realizadas no capítulo 5, podemos começar a abordar as questões de investigação identificadas na secção 1.2. Em relação à

primeira questão, "Será possível criar uma simulação de fluidos interativa num motor de jogos, mantendo o comportamento fisicamente realista e esperado da simulação", criámos diversos cenários de simulação, apresentados na secção 4.2.1. Como foi visto no capítulo 5, ao testar a simulação dos fluidos nos diferentes cenários, na secção 5.3 é possível verificar que os nossos fluidos podem descrever diversos comportamentos que respeitam o previsível comportamento de um fluido real, desde salpicos, ondas ou cascatas. Para além disso, no cenário "Câmara de ondas", verifica-se que quando o fluido interage com outros objetos rígidos, o seu comportamento continua como esperado. Todavia, é importante ressalvar que para que a interação com outros corpos seja computada, ambos os objetos a simular têm de compreender o mesmo passo de tempo.

Para responder à segunda questão, de "Como criar um sistema de simulação de fluidos parameterizáveis, numa cena tridimensional, que possibilite personalização", no capítulo 3 são descritos os diversos requisitos e arquitetura do sistema de fluidos. Por sua vez, na secção 4.2, são apresentados dois controladores cruciais para a criação do fluido e a definição dos seus diferentes parâmetros. Após a implementação do sistema, na secção 5.3.2, conclui-se que as diferentes parametrizações resultam em diversos fluidos, como foi o caso da água e do mel. Para finalizar, em relação à questão de "Quais os melhores algoritmos de simulação de fluidos, gerados por interação de partículas, para reproduzir uma simulação eficiente, realista e adequada ao cenário/comportamento que se pretende simular", com base na avaliação dos resultados obtidos dos testes realizados no capítulo 5, chegou-se a diversas constatações apresentadas na secção 5.4. Dessa análise, conclui-se que a escolha do melhor algoritmo de simulação deve ser tomada com base no cenário de simulação, e que deve ser parametrizado considerando a especificidade do fluido pretendido.

Com base nos resultados obtidos, é possível retirar algumas conclusões. Ao analisar esses resultados, observa-se que um dos principais fatores que influência a escolha do algoritmo/otimização é a quantidade de partículas a simular. Com base nesta métrica, a otimização que obtém melhores resultados, é a otimização com recurso a uma grelha auxiliar, que permite simular uma quantidade superior de partículas mantendo uma frequência de quadros por segundo (FPS) a cima dos 60. Contudo, se a área em que se pretende simular o comportamento do fluido tiver dimensões demasiado elevadas, a criação de uma grelha com ditas dimensões deixa de ser rentável. Como tal, para situações como essa, o algoritmo mais adequado, estará entre a otimização "Simetria de forças"e "Monte-Carlo". Para decidir entre estes dois algoritmos deve ser considerado, o quão menosprezável pode ser a precisão dos resultados, uma vez que "Monte-Carlo"permite que se menosprezem algumas interações, com a finalidade de obter resultados mais rápidos.

6.1 Trabalho Futuro

Refletindo sobre o que foi apresentado no capítulo 2, entende-se que na Simulação de Fluidos, existe uma grande diversidade de propriedades e funcionalidades a implementar.

Como tal, as possibilidades para a investigação de trabalhos futuros são imensas. Nesta secção, serão apresentadas algumas propostas de trabalhos a realizar futuramente.

Uma primeira propriedade que poderia ser interessante implementar como trabalho futuro, seria a Tensão Superficial. A Tensão Superficial é uma propriedade fundamental no comportamento dos líquidos, que surge como impacto das forças entre as moléculas do mesmo. Esta propriedade pode ser pertinente para tornar o fluido simulado muito mais realista, ao comportar uma espécie de membrana na superfície do mesmo. Para além disso, a simulação da superfície faz com que as ondas e comportamentos do fluido ao interagir com obstáculos sejam muito mais precisos e reais.

Como referido na secção 5.3.1, na análise do cenário da Câmara com obstáculos, a nossa implementação não simula os fluidos como o ar ou outros gases, como tal, seria interessante integrar esse tipo de fluidos e assim, possibilitar a criação de um sistema de vasos comunicantes. Ademais, a interação entre fluidos de diferentes consistências pode criar diversos comportamentos interessantes.

Uma última proposta de trabalho futuro, seria aumentar número de partículas que se pode simular, pois pode ser uma contribuição importante para a simulação de fluidos. Deste modo, integrar Computação Paralela ganha alguma relevância. A Computação Paralela é uma técnica frequentemente utilizada para melhorar a velocidade e eficiência das simulações, distribuindo a carga de processamento entre diversos processadores. Investigações futuras podem explorar como é que a computação paralela pode ser usada para simular diversos fluidos em tempo real e/ou para simular fluidos constituídos por quantidades de partículas muito mais elevadas.

Bibliografia

[1] N. Ang, A. Catling, F. C. Ciardi e V. Kozin. "The Technical Art of Sea of Thieves". Em: *ACM SIGGRAPH 2018 Talks*. SIGGRAPH '18. Vancouver, British Columbia, Canada: Association for Computing Machinery, 2018. ISBN: 9781450358200. DOI: 10.1145/3214745.3214820. URL: https://doi.org/10.1145/3214745.3214820 (ver p. 32).

[2] A. Angelidis e F. Neyret. "Simulation of smoke based on vortex filament primitives". Em: *SCA '05*. 2005 (ver p. 21).

[3] M. Atif, S.-W. Chi, E. Grossi e A. Shabana. "Evaluation of breaking wave effects in liquid sloshing problems: ANCF/SPH comparative study". Em: *Nonlinear Dynamics* 97 (jul. de 2019). DOI: 10.1007/s11071-019-04927-5 (ver p. 41).

[4] G. K. Batchelor. *An Introduction to Fluid Dynamics*. Cambridge University Press, 1973 (ver p. 17).

[5] M. Becker e M. Teschner. "Weakly Compressible SPH for Free Surface Flows". Em: vol. 9. Jan. de 2007, pp. 209–217. DOI: 10.1145/1272690.1272719 (ver pp. 22, 28).

[6] C. Braley e A. Sandu. "Fluid Simulation For Computer Graphics: A Tutorial in Grid Based and Particle Based Methods". Em: 2009 (ver pp. 14, 18).

[7] R. Bridson. *Fluid Simulation for Computer Graphics, Second Edition*. Taylor e Francis, 2015. ISBN: 9781482232837. URL: https://books.google.pt/books?id=7MySoAEACAAJ (ver pp. 9–11, 16, 17, 52).

[8] R. Bridson e M. Müller. "Fluid simulation: SIGGRAPH 2007 course notes Video files associated with this course are available from the citation page". Em: (ago. de 2007), pp. 1–81. DOI: 10.1145/1281500.1281681 (ver pp. 23, 28).

[9] N. Chentanez e M. Müller. "Real-Time Eulerian Water Simulation Using a Restricted Tall Cell Grid". Em: *ACM Trans. Graph.* 30.4 (jul. de 2011). ISSN: 0730-0301. DOI: 10.1145/2010324.1964977. URL: https://doi.org/10.1145/2010324.1964977 (ver p. 23).

[10] E. Cook. "Creating the Flood Effects in Uncharted 3". Em: GDCVault, 2012. URL: https://www.gdcvault.com/play/1015723/Creating-the-Flood-Effects-in (ver p. 30).

[11] M. Desbrun e M.-P. Gascuel. "Smoothed particles: a new paradigm for animating highly deformable bodies". Em: 1996 (ver pp. 19, 20, 27, 40, 42).

[12] J. E. Fromm e F. H. Harlow. "Numerical Solution of the Problem of Vortex Street Development". Em: *Physics of Fluids* 6.7 (jul. de 1963), pp. 975–982. DOI: 10.1063/1.1706854 (ver p. 1).

[13] J. Fuentes e F. Luo. "Synchronizing Parallel Geometric Algorithms on Multi-Core Machines". Em: *International Journal of Networking and Computing* 8 (jul. de 2018), pp. 240–253. DOI: 10.15803/ijnc.8.2_240 (ver p. 61).

[14] P. Gac, D. Emmanuelle, P.-Y. Louis e L. Aveneau. "Efficient Simulation of Fluids". Em: jul. de 2019. ISBN: 978-1-78985-327-8. DOI: 10.5772/intechopen.86619 (ver p. 48).

[15] M. N. Gamito, P. Lopes e M. R. Gomes. "Two-dimensional simulation of gaseous phenomena using vortex particles". Em: 1995 (ver p. 21).

[16] R. A. Gentry, R. E. Martin e B. J. Daly. "An Eulerian differencing method for unsteady compressible flow problems". Em: *Journal of Computational Physics* 1.1 (1966), pp. 87–118. ISSN: 0021-9991. DOI: https://doi.org/10.1016/0021-9991(66)90014-3. URL: https://www.sciencedirect.com/science/article/pii/0021999166900143 (ver p. 1).

[17] R. A. Gingold e J. J. Monaghan. "Smoothed particle hydrodynamics: Theory and application to non-spherical stars". Em: *Monthly Notices of the Royal Astronomical Society* 181 (1977), pp. 375–389 (ver p. 21).

[18] F. H. Harlow. "The particle-in-cell method for numerical solution of problems in fluid dynamics". Em: (mar. de 1962). DOI: 10.2172/4769185. URL: https://www.osti.gov/biblio/4769185 (ver p. 1).

[19] F. H. Harlow e J. E. Welch. "Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface". Em: *Physics of Fluids* 8.12 (dez. de 1965), pp. 2182–2189. DOI: 10.1063/1.1761178 (ver p. 1).

[20] E. Hastings e J. Mesit. "Optimization of large-scale, real-time simulations by spatial hashing". Em: (jan. de 2005) (ver p. 53).

[21] D. Hinsinger, F. Neyret e M.-P. Cani. "Interactive Animation of Ocean Waves". Em: *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '02. San Antonio, Texas: Association for Computing Machinery, 2002, pp. 161–166. ISBN: 1581135734. DOI: 10.1145/545261.545288. URL: https://doi.org/10.1145/545261.545288 (ver p. 24).

[22] Z. Horváth e A. Herout. “Real-time particle simulation of fluids”. Em: mai. de 2012 (ver p. 27).

[23] Z. Horváth e A. Herout. “Real-time particle simulation of fluids”. Em: mai. de 2012 (ver p. 53).

[24] S. Jeschke, C. Hafner, N. Chentanez, M. Macklin, M. Müller-Fischer e C. Wojtan. “Making Procedural Water Waves Boundary-Aware”. Em: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Goslar, DEU: Eurographics Association, 2020. URL: https://doi.org/10.1111/cgf.14100 (ver p. 25).

[25] T. Kellomäki. “Fast Water Simulation Methods for Games”. Em: *Comput. Entertain.* 16 (2017), 2:1–2:14 (ver p. 30).

[26] D. Kim. *Fluid Engine Development*. English. 1st. ACM Books. Boca Raton, FL: CRC Press, 2016. 370 pp. ISBN: 9781498719926. URL: https://www.crcpress.com/Fluid-Engine-Development/Kim/p/book/9781498719926 (ver pp. 53–55).

[27] H. Lamb. *Hydrodynamics*. Cambridge university press, 1932 (ver p. 17).

[28] R. Lee e C. O’Sullivan. “A Fast and Compact Solver for the Shallow Water Equations.” Em: vol. 1. Jan. de 2007, pp. 51–57. DOI: 10.2312/PE/vriphys/vriphys07/051-057 (ver pp. 26, 27).

[29] M. Lewis e J. Jacobson. “Game engines in scientific research”. Em: *Communications of the ACM* 45 (jan. de 2002), pp. 27–31 (ver p. 30).

[30] J.-C. Lombardo e C. Puech. “Oriented particles: A tool for shape memory objects modelling”. Em: 1995 (ver p. 20).

[31] L. B. Lucy. “A numerical approach to the testing of the fission hypothesis.” Em: *The Astronomical Journal* 82 (1977), pp. 1013–1024 (ver p. 21).

[32] M. Macklin e M. Müller. “Position Based Fluids”. Em: *ACM Transactions on Graphics* 32 (jul. de 2013), 104:1–. DOI: 10.1145/2461912.2461984 (ver pp. 28, 29).

[33] X. Mei, P. Decaudin e B.-G. Hu. “Fast Hydraulic Erosion Simulation and Visualization on GPU”. Em: *15th Pacific Conference on Computer Graphics and Applications (PG’07)*. 2007, pp. 47–56. DOI: 10.1109/PG.2007.15 (ver p. 32).

[34] M. Müller, D. Charypar e M. Gross. “Particle-Based Fluid Simulation for Interactive Applications”. Em: *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’03. San Diego, California: Eurographics Association, 2003, pp. 154–159. ISBN: 1581136595 (ver pp. 27, 28, 40–43, 50, 70).

[35] R. Nóbrega, A. Sabino, A. Rodrigues e N. Correia. “Flood Emergency Interaction and Visualization System”. Em: *Visual Information Systems. Web-Based Visual Information Search and Management*. Ed. por M. Sebillo, G. Vitiello e G. Schaefer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 68–79. ISBN: 978-3-540-85891-1 (ver p. 55).

[36] NVIDIA. *Particle Simulation with CUDA*. https://developer.download.nvidia.com/assets/cuda/files/particles.pdf. Accessed on March 30, 2023. 2010 (ver pp. 48, 53).

[37] S. I. Park e M.-J. Kim. "Vortex fluid for gaseous phenomena". Em: *SCA '05*. 2005 (ver p. 21).

[38] C. Pozzer, C. De, C. Pahins, I. Heldal, J. Mellin e P. Gustavsson. "A Hash Table Construction Algorithm for Spatial Hashing Based on Linear Memory". Em: vol. 2014. Nov. de 2014. DOI: 10.1145/2663806.2663862 (ver p. 53).

[39] W. T. Reeves. "Particle Systems—a Technique for Modeling a Class of Fuzzy Objects". Em: *ACM Trans. Graph.* 2.2 (abr. de 1983), pp. 91–108. ISSN: 0730-0301. DOI: 10.1145/357318.357320. URL: https://doi.org/10.1145/357318.357320 (ver pp. 18, 19).

[40] D. Rioux-Lavoie, R. Sugimoto, T. Özdemir, N. H. Shimada, C. Batty, D. Nowrouzezahrai e T. Hachisuka. "A Monte Carlo Method for Fluid Simulation". Em: *ACM Trans. Graph.* 41.6 (nov. de 2022). ISSN: 0730-0301. DOI: 10.1145/3550454.3555450. URL: https://doi.org/10.1145/3550454.3555450 (ver p. 51).

[41] B. Solenthaler. "Predictive-corrective incompressible SPH". Em: *ACM Trans. Graph. Article* 28 (set. de 2009). DOI: 10.1145/1576246.1531346 (ver pp. 22, 28).

[42] J. Stam. "Real-Time Fluid Dynamics for Games". Em: (mai. de 2003) (ver p. 8).

[43] J. Stam. "Stable Fluids". Em: *ACM SIGGRAPH 99* 1999 (nov. de 2001). DOI: 10.1145/311535.311548 (ver p. 2).

[44] A. Stomakhin e A. Selle. "Fluxed Animated Boundary Method". Em: *ACM Trans. Graph.* 36.4 (jul. de 2017). ISSN: 0730-0301. DOI: 10.1145/3072959.3073597. URL: https://doi.org/10.1145/3072959.3073597 (ver pp. 13–15).

[45] D. Stora, P.-O. Agliati, M.-P. Cani, F. Neyret e J.-D. Gascuel. "Animating Lava Flows". Em: *Graphics Interface* (dez. de 2000) (ver pp. 20, 21).

[46] R. Szeliski e D. Tonnesen. "Surface modeling with oriented particle systems". Em: *1992 International Conference on Computer Graphics and Interactive Techniques*. ACM, jun. de 1992, pp. 185–194. URL: https://www.microsoft.com/en-us/research/publication/surface-modeling-with-oriented-particle-systems/ (ver p. 20).

[47] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranets e M. Gross. "Optimized Spatial Hashing for Collision Detection of Deformable Objects". Em: *VMV'03: Proceedings of the Vision, Modeling, Visualization* 3 (dez. de 2003) (ver p. 53).

[48] J. Tessendorf. "Simulating Ocean Water". Em: *SIG-GRAPH'99 Course Note* (jan. de 2001) (ver p. 32).

[49] N. Thürey, M. Müller, S. Schirm e M. Gross. "Real-time Breaking Waves for Shallow Water Simulations". Em: out. de 2007, pp. 39–46. DOI: 10.1109/PG.2007.33 (ver p. 27).

[50] R. Valencia-Garcıa, K. Lagos-Ortiz, G. Alcaraz-Mármol, J. del Cioppo e N. Vera-Lucio, eds. *Technologies and Innovation - Second International Conference, CITI 2016, Guayaquil, Ecuador, November 23-25, 2016, Proceedings*. Vol. 658. Communications in Computer and Information Science. 2016. ISBN: 978-3-319-48023-7. DOI: 10.1007/978-3-319-48024-4. URL: https://doi.org/10.1007/978-3-319-48024-4 (ver p. 29).

[51] A. P. Witkin e P. S. Heckbert. "Using Particles to Sample and Control Implicit Surfaces". Em: *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '94. New York, NY, USA: Association for Computing Machinery, 1994, pp. 269–277. ISBN: 0897916670. DOI: 10.1145/192161.192227. URL: https://doi.org/10.1145/192161.192227 (ver p. 20).

[52] J. Wolper, M. Gao, M. Lüthi, V. Heller, A. Vieli, C. Jiang e J. Gaume. "A glacier–ocean interaction model for tsunami genesis due to iceberg calving". Em: *Communications Earth and Environment* 2 (jun. de 2021). DOI: 10.1038/s43247-021-00179-7 (ver pp. 1, 2).

[53] L. Yaeger, C. Upson e R. Myers. "Combining Physical and Visual Simulation—Creation of the Planet Jupiter for the Film 2010". Em: *SIGGRAPH Comput. Graph.* 20.4 (ago. de 1986), pp. 85–93. ISSN: 0097-8930. DOI: 10.1145/15886.15895. URL: https://doi.org/10.1145/15886.15895 (ver p. 21).